# Modélisation et résolution de problèmes difficiles de transport à la demande et de Lot-Sizing

Samuel Deleplanque

Numéro d'ordre : D.U.
EDSPIC :

**Université Blaise Pascal - Clermont-Ferrand II**

**Ecole doctorale : Sciences Pour l'Ingénieur de Clermont-Ferrand**

**Thèse**

Présentée par

**Samuel DELEPLANQUE**

pour obtenir le grade de

**Docteur d'Université**

Spécialité : Informatique

# Modélisation et résolution de problèmes difficiles de transport à la demande et de *Lot-Sizing*

Soutenue publiquement le 12 Septembre 2014 devant le jury composé de :

| | | |
|---|---|---|
| *Président :* | Dominique FEILLET | - Prof. aux Mines, Saint-Etienne |
| *Rapporteur :* | Aziz MOUKRIM | - Prof. à l'UT, Compiègne |
| *Directeurs de thèse :* | Alain QUILLIOT | - Prof. à l'UBP, Clermont-Ferrand II |
| | Jean-Pierre DERUTIN | - Prof. à l'UBP, Clermont-Ferrand II |

*« La connaissance du chemin ne peut pas se substituer au fait de mettre un pied devant l'autre. »*

*Mary Caroline Richards*

## Remerciements

Je tiens à adresser mes plus sincères remerciements à Messieurs Alain Quilliot et Jean-Pierre Dérutin pour leurs judicieux conseils, leur apport critique et leur direction attentive au cours de ces années de recherche. Je remercie à nouveau M. Quilliot pour son accueil au sein du LIMOS (Informatique, Modélisation et Optimisation des Systèmes, UMR CNRS 6158) qu'il dirige : le dynamisme du laboratoire et la motivation de ses membres m'ont permis de travailler avec, j'espère, efficacité et dans un climat de confiance. Malgré la direction du laboratoire, M. Quilliot a su trouver le temps pour me soutenir.

Je tiens également à exprimer ma profonde reconnaissance à tous les étudiants, techniciens, ingénieurs et chercheurs que j'ai côtoyés le long de ces trois années et en particulier à :

- Madame Safia Kedad-Sidhoum avec laquelle il fut très agréable de travailler sur le Lot-Sizing. Elle m'accueillit plusieurs fois au LIP6, m'encouragea à ma première présentation en anglais à Oxford et m'aida à la rédaction d'un premier article ;
- Ana Luísa Marques et Luiza Bernardes Real que j'ai "tuteurées" lors de leur projet et de leur stage d'ingénieurs ; mais aussi Benoit Bernay, ingénieur au LIMOS. Ils ont tous les trois travaillé avec un grand sérieux sur le problème de la minimisation du nombre d'arrêts ;
- Messieurs Philippe Vaslin et Marc Davis, le premier, maître de conférence au LIMOS et le second, son doctorant qui réalisa sa thèse en même temps que moi. Nous avons tous les trois travaillé dans une excellente ambiance au sein du même bureau à l'ISIMA ;
- Hélène Toussaint, ingénieur de recherche, dont j'ai pu apprécier la gentillesse, l'expérience et la compétence ;
- Khouloud Boukadi, aujourd'hui maître de conférence à Sfax - Tunisie -, qui m'a longtemps fait partagé son expérience de doctorante à l'école des mines de Saint-Etienne lors de son ATER au LIMOS. Ce sont nos échanges qui m'ont motivé pour faire de même ;

Je n'oublie pas non plus les autres membres du laboratoire qui rendent la vie quotidienne très agréable (Claude, Nicolas, Béa, M. Hou et bien d'autres) ;

Enfin, je ne peux qu'avoir une pensée pour mes parents et mon frère : tous les trois ont su être présents, attentifs et m'épauler notamment pour la relecture de ce mémoire.

# Table des matières

# Table des figures

# Liste des tableaux

# Liste des algorithmes

# Introduction Générale

Bien des personnes associent le mot transport à l'utilisation d'une voiture individuelle. C'est le cas, d'ailleurs, dans 90% des déplacements, avec la plupart du temps une seule personne à bord. Aux problèmes de circulation que l'on imagine (ou plutôt que l'on vit au quotidien) s'ajoutent ceux des accidents - le risque s'accroît avec le nombre de véhicules -, de la pollution, bien sûr, de la prolifération des parkings (jamais en nombre suffisant pourtant) et de bien d'autres nuisances.

De façon générale la gestion des flux dans les grandes villes est devenue très difficile. La situation déjà grave que nous connaissons devient critique, pour ne pas dire apocalyptique, dans certaines métropoles d'Asie par exemple, ou en cas d'événement exceptionnel. Aujourd'hui, de nouvelles organisations et mutualisations de véhicules sont nécessaires. Ces besoins apparaissent à la fois en zones rurales et urbaines, pour de multiples raisons : le vieillissement de la population entraînant des problèmes de dépendance, la désertification des campagnes et la diminution des services assurés autrefois, l'augmentation des coûts énergétiques, les difficultés de la circulation, etc...

Le problème, on le voit, est important et délicat : il ne fera que s'intensifier. L'accroissement de la population urbaine et péri-urbaine et la mobilité de plus en plus grande des personnes, due au contexte technique, économique et social, ont rendu obligatoire, dans le cadre de différentes politiques d'aménagement du territoire, la remise en cause de la prééminence des transports individuels au profit de ceux qui mutualisent l'utilisation des véhicules. Ceci étant, la plupart des transports publics proposés aujourd'hui obéissent à des règles manquant de souplesse et n'incluent que rarement le caractère dynamique, en temps et en espace, de la demande, réduisant ainsi l'attractivité de ces services et les rendant même parfois difficilement supportables. De ce fait, la majorité des usagers utilisent encore leur propre véhicule. La littérature dans le domaine de l'urbanisation et de la géomatique nous indique que des transports à la demande performants remplaçant pour une grande part les véhicules privés pourraient être la clé d'une circulation urbaine harmonieuse et agréable. L'avancée technologique des véhicules sans conducteur et des moyens de communication devrait être un tremplin pour ces transports dynamiques.

Le management de ces nouveaux flux tend à se faire à l'aide des TIC (géolocalisation, communications mobiles, web services). Aussi, les modèles de la Recherche Opérationnelle (RO) peut contribuer à optimiser le rapport coût/qualité de service (QoS). Cette notion de compromis est essentielle dans la gestion des systèmes

émergents. Alors qu'auparavant seuls les coûts étaient pris en compte (coûts de production, coûts de fonctionnement d'une flotte de véhicules), la performance d'une solution à un tel problème se mesure aujourd'hui également à la qualité du service rendu (temps d'acheminement des usagers d'un service de transport), à son empreinte écologique, etc...

Le principal objet de cet thèse réside dans la modélisation et l'optimisation de services de transport à la demande aussi différents soient-ils (ou seront-ils). Les techniques de supervision doivent alors pouvoir supporter différents objectifs et différentes contraintes pour s'adapter aux services actuels et futurs. Ainsi, ce rapport de thèse développe différentes variantes du DARP - ang. *Dial-a-Ride Problem* -, le problème de RO modélisant et optimisant un service classique de transport à la demande. Le DARP standard a été étendu de façon à prendre en compte des hypothèses de fonctionnement prometteuses, comme le fait de séparer les composants d'une même requête pour les dispatcher sur des véhicules différents ou encore la présence de mécanismes d'intermodalité (cf. figure 1).

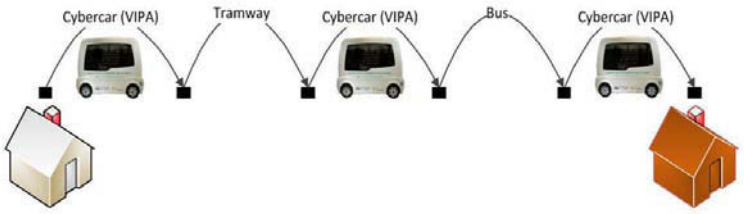


FIGURE 1 – Intégration du VIPA dans le transport intermodal

Le passage d'un service à l'autre se fait par un transfert (transbordement). Les contextes statiques (toutes requêtes connues en amont de la résolution) et dynamiques (les demandes sont intégrées en cours de tournée) doivent également être examinés. De ce fait, nous adoptons ici le point de vue dominant des algorithmes dits d'insertion, permettant d'articuler statique et dynamique. La littérature scientifique sur le DARP dynamique souffre d'une réelle lacune dès lors que l'on étudie un contexte qui peut-être qualifié de *dynamique pure* où les tournées courantes deviennent de moins en moins flexibles au fil du temps.

Aussi, cette thèse permet-elle d'inscrire les véhicules autonomes tels que les VIPA dans de nouvelles problématiques de la Recherche Opérationnelle tout en restant dans le domaine du transport à la demande. La modélisation puis l'optimisation de ces systèmes permet de créer les plannings de ces nouveaux véhicules. A long terme, l'évolution technologique devrait permettre de ne plus se soucier du fait qu'ils sont automatiques. Ces travaux tentent de fournir un cadre suffisamment générique permettant à la fois de fournir une solution exploitable aujourd'hui et qui soit adaptable demain.

Les travaux présentés dans ce mémoire se classent en trois catégories. La première, recouvrant 2 chapitres, étudie les problèmes de transport à la demande selon deux modes de fonctionnement bien spécifiques : les transferts et la division du chargement. La seconde, sur 3 chapitres, concerne les différents critères de qualité recher-

chés : rapport Coût/QoS, Fiabilité (véhicules sans conducteur) et enfin Robustesse (DARP dynamique). La thèse se conclut sur une question de production incluant un problème de transport, plus précisément un cas particulier du problème de *Lot-Sizing*. Ce problème est multi-produits, multi-sites, multi-périodes, multi-capacités, et intègre des coûts fixes de production. Avant ces 5 chapitres sont présentés le contexte général dans lequel ces travaux de recherche interviennent, puis l'état de l'art sur la modélisation et l'optimisation des différents problèmes de transports étudiés dans ce mémoire.

La première étude sur les transports à la demande débute donc au chapitre 3, elle met au point une résolution heuristique par insertions particulièrement rapide du problème de *Dial-a-Ride* (DARP) qui pourra être aisément déployé sur un système multi-processeurs et adapté à une exploitation concrète dynamique. Elle se concentre sur les contraintes temporelles en réduisant l'espace solution par la contraction des fenêtres de temps selon l'enchaînement des nœuds de chaque tournée. De nombreuses expériences recherchant différents critères de performance appuient ce travail et le tout a fait l'objet de conférences, internationale (avec actes) et nationale ([Deleplanque and Quilliot (2012b)] et [Deleplanque et al. (2012d)]).

Le chapitre 4 poursuit sur le DARP en l'ouvrant à la possibilité de diviser le chargement pour permettre, par exemple, l'acheminement d'un grand groupe de personnes. Nous avons repris le modèle et les techniques utilisés dans le chapitre précédent. Comme c'est la première étude qui s'appuie sur cette hypothèse, nous lui avons donné un nom : *Dial-a-Ride Problem with Split Loads* (DARPSL). La propagation de contraintes est utilisée pour tester la non violation des contraintes de temps qui, pour certaines, se démultiplient autant de fois que les demandes sont partagées. Lors des expérimentations, la génération d'instances a paru nécessaire après une relecture de l'état de l'art du DARP pour exécuter nos algorithmes sur des scénarios bien précis. Ces travaux ont fait l'objet d'un article dans *Studies in Computational Intelligence* édité par *Springer* ([Deleplanque and Quilliot (2013a)]).

Le chapitre 5 introduit lui aussi une nouvelle variante du DARP : le *Dial-a-Ride with transfers* (DARPT) peu étudié. Il permet l'utilisation d'un transbordement dans la satisfaction d'une demande. Nous le résolvons toujours par insertions successives dont la validité est établie par propagation de contraintes dont celles de précédence liées aux transferts. En effet, le processus de propagation s'élargit ici à plusieurs tournées lorsque la satisfaction d'une requête est testée. Notre solution heuristique est à nouveau mise à l'épreuve par des instances que nous avons générées selon un scénario bien précis à base de *clustering*. Ici, nous disposons d'un ensemble de sous-flottes provenant de dépôts différents et autant de sous-régions. De manière empirique, les transbordements se font, pour la plupart, aux frontières de ces sous-régions. Ces travaux ont été présentés en France et à l'étranger puis publiés ([Deleplanque and Quilliot (2013b)], [Deleplanque and Quilliot (2013c)] et [Deleplanque and Quilliot (2013f)]).

Le chapitre suivant introduit un traitement spécifique de la notion de robustesse pour le cas général du DARP dynamique. Nous établissons une mesure d'*Insérabilité* qui permet d'évaluer l'impact d'une insertion au sein d'une tournée sur la place

laissée aux futures demandes. Nous expliquons également comment représenter les demandes futures (notion de demandes virtuelles) et mettre en place une simulation afin de tester l'ensemble de nos algorithmes dans un contexte dynamique. Ici, la résolution par insertions successives prend tout son sens. En effet, l'heuristique s'adapte au dynamique sans grosse difficulté et supporte également l'intégration d'une notion de robustesse. Contrairement au DARP dynamique de la littérature, nous prenons en compte la synchronisation véhicule-usager (rendez-vous) réduisant considérablement la flexibilité des tournées courantes à la venue prochaine d'autres demandes. Plusieurs communications nationales et internationales (article, chapitre et conférences) ont été faites pour cette étude ([Deleplanque and Quilliot (2014b)], [Deleplanque and Quilliot (2014c)], [Deleplanque and Quilliot (2014a)], [Deleplanque and Quilliot (2013d)], [Deleplanque et al. (2013a)] et [Deleplanque and Quilliot (2013e)]).

Le chapitre 7 traite d'une variante du DARP lorsqu'il est confronté aux contraintes relatives à l'autonomie des véhicules (VIPA, Cycab etc.) et que ces derniers sont déployés sur circuit. Ainsi, la topologie (sur circuit ou en ligne) et les contraintes de fiabilité nous amènent-elles à minimiser le nombre d'arrêts effectués par la flotte de véhicules : nous essayons de résoudre le problème en ligne et de manière exacte par un *Branch-and-Cut-and-Price* incluant un problème de flots et une première utilisation de la relaxation lagrangienne. Nous étudions et testons également la mise en perspective de ce problème grâce au DARP en y incluant des fenêtres de temps et des durées maximum d'acheminement. Le sujet a été présenté en conférence (nationale et internationale) ([Deleplanque et al. (2013c)] et [Quilliot et al. (2014)]).

Toujours dans le cadre de ce doctorat, d'autres communications, plus générales sur les problèmes de transport à la demande, ont été réalisées ([Deleplanque (2012)], [Deleplanque (2013)]) et enfin plusieurs articles sont en cours de relecture.

Enfin, avant de conclure, le dernier chapitre reprend les outils que sont la modélisation en problèmes de flots et la relaxation lagrangienne afin de résoudre un problème couplant transport et production. Ce problème particulier de *Lot-Sizing* consiste à essayer de satisfaire un ensemble de demandes qui se caractérisent par un type de produit et sa quantité espérée dans l'espace et dans le temps. La performance d'une solution satisfaisant l'ensemble de ces demandes est mesurée selon la somme des coûts totaux de la production (variables et fixes), du stockage et du transport dans les différentes périodes. Il est particulièrement difficile d'en donner la solution optimale puisqu'il inclut, en plus des limites par usine sur les quantités produites, des capacités sur le stockage et sur les transferts de produits inter-usines. Nous le résolvons par une décomposition en deux problèmes : un premier (maître) de localisation intégrant une relaxation lagrangienne et un second (esclave) de multi-flots. Le premier permet de s'affranchir des contraintes de capacité et le second celles de *setup* (qui peuvent être vues comme la préparation des machines à la production et qui engendrent un coût fixe). Ce travail a fait l'objet de plusieurs conférences nationales et internationales dont certaines avec actes ([Deleplanque and Quilliot (2012a)], [Deleplanque et al. (2012a)], [Deleplanque et al. (2012c)] et [Deleplanque et al. (2012b)]). Un article a également été publié dans *RAIRO Operation Research* [Deleplanque et al. (2013b)].

# Chapitre 1

# Généralités sur l'algorithmique et la Recherche Opérationnelle

## 1.1 Algorithmique et notion de Complexité

L'algorithmique et la théorie de la complexité vont de pair. En effet, le but de la Complexité est de proposer un outil mesurant a priori la difficulté d'un problème et les performances d'un algorithme. "A priori" signifie que cette évaluation se fait avant l'implémentation, indépendamment du langage des couches logiciels utilisées afin de mieux cerner la difficulté du problème et permettre ainsi d'aider au choix d'une solution de résolution. Il faut alors distinguer cette notion d'évaluation de celle qui, a posteriori, est réalisée au moyen de tests et par l'étude des résultats obtenus.

### 1.1.1 Généralités sur les algorithmes

Un algorithme est généralement mis au point pour implémenter la résolution d'un problème composé d'une entrée et d'une sortie (respectivement INPUT et OUTPUT), puis d'un ensemble de relations formelles ou informelles entre les composants de l'INPUT et de l'OUTPUT. Certains systèmes permettent aux éléments de l'OUTPUT de supporter un certain niveau d'erreur. La performance d'un algorithme se mesure selon 4 paradigmes :

- la précision : évaluation de l'erreur de l'OUTPUT par rapport à ce qui était attendu dans la spécification du problème,
- la robustesse : adaptabilité de l'algorithme face aux variations de l'INPUT tel qu'il est spécifié dans le problème,
- le temps d'exécution : nombre d'instructions (classées par type) contenues dans l'algorithme. Cette méthode permet de s'abstraire des comparaisons matérielles (processeur, caches, adressage etc.) ;
- l'espace mémoire : quantité de mémoire nécessaire pour l'exécution de l'algorithme. La mesure est exprimée en espaces mémoires élémentaires. Ce dernier paradigme peut sembler accessoire aujourd'hui en raison de l'importante augmentation de la mémoire vive (RAM) dans les machines de ces dernières années. Ceci étant, les quantités de données traitées dans certains des pro-

blèmes actuels sont très grandes (phylogénétique, simulations météorologiques etc.) et ce paradigme reste dans ce cas une évaluation non négligeable pour les algorithmes qui les résolvent.

C'est le contexte dans lequel le problème doit être résolu qui établit l'importance de chaque paradigme. Un problème dynamique d'établissement des plannings d'une flotte de taxis peut requérir par exemple une grande réactivité (temps d'exécution courts) au détriment parfois de la précision (on ne trouve pas systématiquement les plannings optimaux). Au contraire, les plannings statiques d'une flotte de bus d'un transport en commun plus classique pourront être construits avec une meilleure précision, les temps d'exécution pouvant être plus longs.

Chacun de ces paradigmes permet de définir des mesures de performance au sens du pire cas ou au sens de la moyenne. Par exemple, la complexité-temps définie au pire cas d'un algorithme mesure le plus grand nombre d'instructions (temps d'exécution). Beaucoup considèrent aujourd'hui les mesures de la complexité définies au pire cas comme peu réalistes. En effet, le pire cas n'est que très rarement atteint et les instances qui les forment sont souvent particulières. Certains algorithmes comme le simplexe peut résoudre un programme linéaire à variables continues dans un nombre polynomial d'instructions mais il existe de très rares instances résolues en $2^{n+m}$ avec $n$ et $m$ nombres de lignes puis de colonnes. Les difficultés d'évaluer à priori la performance d'un algorithme nous amènent généralement à le faire à posteriori par le biais de procédures de tests. Cependant, d'autres difficultés apparaissent et sont liées à la génération d'instances du problème. Il est connu que que pour la plupart des problèmes à réponse binaire (existence (ou pas) d'un chemin hamiltonien, coloration possible (ou pas) d'un graphe avec $k$ couleurs etc...) dont les instances ont été produites par un processus de génération (apparemment neutre) de nombres pseudo-aléatoires, ces derniers vont induire une convergence forte de la réponse au problème vers une des valeurs possibles 0 ou 1. Ceci signifie qu'il est très difficile de générer des instances, pour un problème donné, qui n'aient pas de propriétés cachées susceptibles d'impacter la performance des algorithmes.

#### Cas des algorithmes dynamiques

Un algorithme *online* ou dynamique $AD$ est appelé successivement pour le traitement d'un flux de données. Chaque appel de $AD$ agit sur la trajectoire du système. Un tel algorithme dynamique est couplé avec un module que l'on notera $DEC$ et qui détermine le moment $t$ où doit être exécuté $AD$. L'évolution du système est commandée à la fois par l'action du couple ($AD$, $DEC$) et par les évènements extérieurs. La supervision des véhicules d'un service de transport à la demande a en principe vocation à s'effectuer à l'aide d'un algorithme de décision dynamique. Alors que les véhicules sont déjà en train de satisfaire diverses requêtes, de nouveaux usagers intègrent le système en formant de nouvelles demandes, que le module de décision doit affecter aux véhicules. Les performances d'un tel algorithme sont difficiles à mesurer, car elles concernent les trajectoires prises par le système. Elles doivent être évaluées par simulation.

### 1.1.2 Les problèmes et leur Complexité

La Théorie de la Complexité a aussi comme objectif d'établir une hiérarchie des problèmes basée sur leur difficulté (lire [Cook (1971)]) et d'identifier des problèmes dominants.

Le premier niveau se situe autour de la notion de décidabilité. Un problème est dit *décidable* (par opposition aux problèmes *indécidables*) s'il existe un algorithme le résolvant, c'est à dire permettant, à partir des données de l'INPUT, d'obtenir l'OUTPUT tel qu'il est décrit dans la spécification du problème. La plupart des problèmes d'Intelligence Artificielle ou de vérification de programmes sont *indécidables*. Les problèmes *indécidables* et *décidables* sont à leur tour partitionnés en d'autres catégories. Ce mémoire ne traite que de problèmes *décidables* : nous faisons donc apparaître leurs principales subdivisions comme les problèmes *P-Temps* et les problèmes *P-Espace* qui, respectivement, sont tels que s'il existe un algorithme polynomial en temps et en espace (chacun pour le pire cas) résolvant (sans erreur) le problème.

Ensuite, la classe *NP-Temps* regroupe les problèmes pour lesquels il est possible d'obtenir un algorithme qui les résout de manière polynomiale. Plus simplement, on peut considérer un problème comme *NP-Temps* si le problème de vérification de ses solutions est *P-Temps*. Cette catégorie est fondamentale puisqu'elle recouvre la plupart des problèmes d'optimisation combinatoire et, de façon plus générale, les problèmes de développement que l'on traite en ingénierie. Elle contient naturellement les problèmes *P-Temps*. Ceci nous permet d'évoquer la célèbre conjecture P≠NP, à savoir qu'il existerait des problèmes *NP-Temps* qui ne sont pas polynomiaux. La réponse est encore ouverte ; par ailleurs, l'institut Clay Mathemics propose encore aujourd'hui un million de dollars ([Clay (2000)]) à celui qui saura prouver l'inégalité ou l'égalité entre les classes P et NP (lire [Fortnow (2009)] pour plus d'informations).

Un des paradigmes les plus célèbres de l'informatique est celui de l'*Universalité*. Il établit une relation de dominance d'un problème sur un autre : un problème P domine un second P' au sens de *NP-Temps*, ou (formulé autrement) : P' est réductible *polynomialement* en P, s'il existe deux algorithmes polynomiaux-temps A1 et A2 tels que :

— A1 construit une entrée de P à partir d'une entrée de P' ;
— A2 construit une sortie de P' à partir d'une sortie de P ;
— le problème P' est satisfait ssi P est satisfait.

**Remarque sur la décomposition/réduction d'un problème difficile.** Un problème difficile pourra se décomposer en plusieurs sous-problèmes, comme le *facility location problem* et le problème de multiflots décomposant celui du *Lot-Sizing* étudié en chapitre 8. Ceci permet une recherche de solutions plus efficace, la littérature disponible étant plus complète pour ces sous-problèmes.

## 1.2 Programmes linéaires et graphes

La **programmation linéaire** constitue un exemple important de problèmes dominants dans la classe *NP-Temps*. Résoudre un programme linéaire consiste à calculer un vecteur de nombres, entiers ou fractionnaires, soumis à un ensemble d'inégalités linéaires et optimisant une fonction également linéaire.

Pour les problème à variables continues, l'algorithme du Simplex, établi au milieu du XX$^{\text{ième}}$ par l'américain George Dantzig (plus de détails dans [Dantzig (1982)]), trouve une solution réalisable (i.e. qui respecte l'ensemble des contraintes) et qui améliore pas à pas cette solution initiale vers la solution optimale. Cette méthode fonctionne bien dans la grande majorité des problèmes posés. Ceci étant, les pires cas rendent le Simplex exponentiel.

Le domaine réalisable d'un programme linéaire en variables continues est appelé **polyèdre**. Pour les programmes linéaires avec des variables entières (PLNE) on ne dispose pas d'un algorithme aussi efficace et on a recours à des procédés tels que le *Branch-and-Bound* ou le *Brand-and-Cut*. Un très grand nombre de problèmes sont modélisables en ces programmes. En effet, par exemple, les variables booléennes permettent directement de modéliser une décision concrète (ouverture ou pas d'une usine, intégration ou pas d'une demande à un véhicule etc.), et c'est ce lien entre variables logiques et variables entières qui explique le grand pouvoir d'expressivité de la PLNE.

#### Outils logiciels PLNE

Plusieurs solutions de ce type existent : CPLEX, XPRESS, LINDO, OSL et bien d'autres. Ils permettent donc de résoudre des PLNE. Ces programmes sont souvent de grande taille. Par exemple, pour le problème de coloration de graphe, un graphe à 1000 sommets et 10000 arêtes peut donner, suivant le PL donné dans l' exemple suivant, va engendrer des millions de variables et de contraintes.

Toujours pour résoudre ces problèmes, malgré les difficultés dues aux limites de mémoire et de temps, on voit apparaître des programmes exécutables maîtres souvent complexes (la plupart en C ou C++) faisant appel à ces bibliothèques mais dans une moindre mesure. Des techniques telles que la génération de colonnes, le *Branch-and-Price* ou le *Branch-and-Cut* permettront d'arriver à une solution dans un temps plus raisonnable.

**Exemple : Problème de coloration de graphes**

Soit les entrées suivantes : $G$ un graphe, $X$ ses nœuds et $E$ ses arêtes. Le problème d'affectation d'une couleur à chaque sommet du graphe de telle sorte que tout couple de sommets adjacents (i.e. une arête les relie) soit coloré différemment et que le nombre de couleurs soit minimal se traduit par le programme linéaire suivant :

- Vecteurs inconnus :
  - $z = z_{x,i}$ avec $i = 1..Card(X), x \in X$ ;
  - T = nombre de couleurs utilisées ;
  - sémantique de $z$ : $z_{x,i} = 1$ ssi $x$ porte la couleur $i$, i.e. $n(x) = i$ ;
- Contraintes :
  - pour toute arête $e = (x, y)$ dans $E$, et toute couleur $i$, $z_{x,i} + z_{y,i} \leq 1$ ;
  - pour tout $x \in X$, $\sum_{i=1..Card(X)} z_{x,i} = 1$ ;
  - pour tout $x \in X$, $\sum_{i=1..Card(X)} i.z_{x,i} \leq T$ ;
- Objectif : Minimiser $T$ ;

La figure 1.1 donne un exemple de coloration d'un graphe de Petersen.

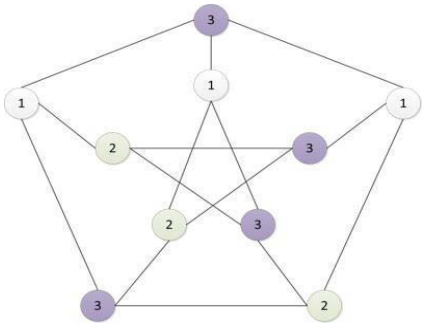


FIGURE 1.1 – Graphe de Petersen au nombre chromatique de 3

Comme la programmation linéaire, la **théorie des graphes** permet de modéliser beaucoup de problèmes d'optimisation combinatoire. Il est d'ailleurs fréquent de transformer un problème de graphe en un programme linéaire et inversement. Il semblerait que l'histoire de cette théorie remonte au XVIII$^{\text{ième}}$ siècle et aux travaux d'Euler. Le fameux problème des 7 ponts de Königsberg (cf. Le graphe de la Figure 1.2) date de la même époque et en est donc une des premières applications. Le principe est simple : les habitants de cette ville désiraient réaliser un circuit les faisant passer une et une seule fois sur chaque pont.

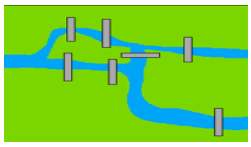

FIGURE 1.2 – Les 7 ponts de Königsberg

Un graphe est un ensemble de nœuds qui sont reliés entre eux par des arcs. Mathématiquement, un graphe est très généralement représenté par un couple de deux ensembles G=(X ;U) où X est l'ensemble des nœuds et U l'ensemble des arcs. Les graphes peuvent être utilisés à titre d'outils, permettant par exemple de modéliser des dépendances entre différents objets ou évènements.

**Cas des flots et multiflots.** A l'intersection entre graphes et programmation linéaire, les modèles de flots/multiflots s'avèrent très utilisés en planification de production et de transport. Un flot est un vecteur indexé sur les arcs d'un graphe orienté qui satisfait la loi de Kirchhoff, c'est-à-dire la loi de conservation des flux qui exprime le fait que les flots entrant dans un nœud sont égaux aux flots sortants. Les arcs peuvent être munis de coûts et de capacités.

Les problèmes les plus classiques de la théorie des flots, et dont le traitement peut se faire en temps polynomial, sont :

— problème du flot maximum : faire circuler un maximum de flot depuis une source vers un puit en respectant des contraintes de capacité ;
— problème du flot à coût minimum : construire un flot soumis à des capacités et minimisant un coût linéaire.

La figure 1.3 est un exemple de problème de *Lot-Sizing* reformulé en problème de flots à coût minimum.

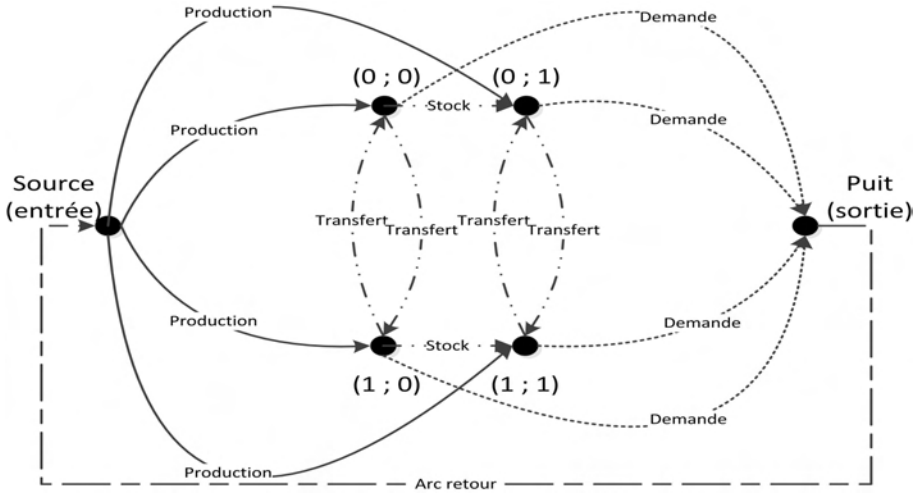


FIGURE 1.3 – Formulation d'un problème de *Lot-Sizing* en un problème de flots

## 1.3 Principaux schémas algorithmiques

### 1.3.1 Algorithmes gloutons et processus de Monte Carlo

Ces algorithmes sont parmi les plus simples mais aussi les plus rapides. L'objet solution $z$ est créé d'une traite à partir d'un objet vide et par un enchaînement de décisions successives qui ne sont jamais remises en cause. Le schéma algorithmique 1 reprend les principaux composants d'un algorithme glouton à destination d'un problème $P$ qui sont :

— l'espace $S'$ des objets partiels dans lequel se développe $z$. $S'$ contient donc un objet vide mais aussi un objet noté $Fail$ (inconsistance). Il inclut également l'espace des objets complets où devrait se trouver $z$ en fin d'algorithme (si succès). Formellement, un objet de S' est une famille de contraintes Ad Hoc.

— l'opérateur de construction $C$ qui agit par adresse sur $z$ dans $S'$. Si $z$ n'est pas complet, $C$ met en jeu un paramètre valeur $u$ afin de restreindre le domaine de liberté de $z$.

— une première fonction booléenne $f1$ de paramètres $(z, u)$ qui permettra de guider le choix du paramètre $u$ en indiquant si $z$ garde ses chances d'être complété en un objet de $S$, c'est à dire, qui respecte l'ensemble des contraintes ;

— une seconde fonction $f2$ également de paramètres $(z, u)$ et renvoyant (généralement) un réel ou entier pour caractériser l'évolution probable de la performance. Elle permet donc d'aider au choix de $u$ afin que $z$ évolue selon la meilleure performance.

**Algorithme 1** Glouton

```
1: z ← NIL ; continu ← VRAI ;
2: Tant que z hors S ET continu Faire
3:    calculer u tel que f1(z,u) ET que f2(z,u) soit maximisé ;
4:    Si la recherche de u est un échec Alors
5:       continu ← FAUX ;
6:    Sinon
7:       appliquer C(z,u) à z ;
8:    Fin si
9: Fin tant que
```

Pour illustrer ces points, prenons l'exemple du problème du voyageur de commerce symétrique (en termes de distances) sur $n$ points :

— l'espace des solutions réalisables $S$ est fait de listes circulaires et contenant une et une seule fois chaque ville $i = 1..n$. Les objets $S'$ seront alors des listes circulaires contenant au plus une fois chaque ville.. Les objets partiels représenteront l'ordre d'enchaînement imposé à certaines villes qui seront éventuellement séparées dans la solution complète par de nouvelles ne figurant pas encore dans la liste ;

— l'opérateur de construction $C$ appliqué à l'objet partiel (liste circulaire) $z$ sélectionne une ville $i$ de $1..n$ n'appartenant pas encore à $z$ puis une ville $j$ déjà dans la liste $z$. Il insère enfin $i$ entre $j$ et $Succ(z, j)$. Le paramètre $u$ est

ici le couple $(i, j)$ ;
— la fonction booléenne $f1$ renvoie alors toujours $VRAI$ ;
— $f2$ retourne l'augmentation du coût DIST(j,i) + DIST(i, Succ(z,j)) - DIST(j, Succ(z,j)), avec DIST(i,j) une fonction de distance entre les villes i et j.

L'approche gloutonne est souvent rendue non déterministe pour que l'on puisse lui appliquer un processus de Monte Carlo tel qu'il est écrit en algorithme 2. Le caractère non déterministe peut s'obtenir dès lors qu'une sélection aléatoire est réalisée sur les meilleurs paramètres $u$ (et non pas sélection systématique du meilleur).

**Algorithme 2** Monte Carlo

1: **Pour** i=1..N **Faire**
2: appliquer l'algorithme Glouton au problème P ;
3: **Fin pour**
4: récupérer la meilleure solution des N réplications ;

Outre le fait qu'ils s'avèrent très efficaces, les processus de Monte Carlo peuvent être parallélisés tant que les différentes réplications restent indépendantes. Les architectures parallèles peuvent lancer chacune des N réplications de l'algorithme Glouton sur un nœud de calcul différent.

### 1.3.2 Algorithmes de "Transformations Locales"

Les algorithmes de "Transformations Locales" agissent sur un objet courant $z$ par applications successives d'un (ou plusieurs) opérateur $T$. Ce dernier est appliqué à $z$ par adresse, via un paramètre de contrôle $u$. L'objet $z$ de départ est généralement obtenu par un algorithme glouton. Si l'objet initial $z$ est généralement une solution réalisable du problème de départ, l'application de $T$ à $z$ peut éventuellement violer les contraintes ou dégrader la qualité de l'objet. Nous retrouvons donc notre fonction $f1$ qui va tester l'impact de l'application de $T$ sur les contraintes pesant sur $z$. Nous retrouvons aussi la fonction $f2$ qui mesure l'impact de $T$ sur la performance de $z$.

L'application d'un algorithme de transformation locale peut être vue comme induisant une trajectoire ou un cheminement de l'objet $z$ courant dans l'espace $S$. Un tel algorithme est dès lors spécifié par son schéma de contrôle, qui fixe les instructions 5 et 6 de l'algorithme 3.

**Algorithme 3** Transformation Locale

1: générer une solution initiale $z$ (e.g. par un algorithme glouton) ;
2: **Tant que** *Continu* **Faire**
3: générer une valeur du paramètre $u$ pour l'opérateur $T$ ;
4: évaluer $f1(z, u)$ et $f2(z, u)$ ;
5: appliquer ou non $T$ suivant les valeurs retournées par $f1$ et $f2$ ;
6: mettre à jour *Continu* ;
7: **Fin tant que**

Il existe une multitude de schémas de contrôle. Il est possible de les répartir en deux catégories : ceux qui acceptent la sélection d'une solution courante $z$ dont la

qualité s'est vue dégradée (selon les critères de performance) et les schémas permettant la sélection d'une solution non réalisable (certaines contraintes ont fait l'objet d'une relaxation).

Parmi les schémas focalisés sur la qualité de la solution, la **Descente** (cf. algorithme 4 et [Hertz and Widmer (2003)]) maintient l'objet dans le domaine réalisable (toutes les contraintes doivent être satisfaites) et n'accepte une nouvelle solution que si celle-ci améliore $z$. Il arrive que l'exécution d'une Descente converge systématiquement sur une même solution finale, qui est alors un optimum global : c'est le cas pour l'optimisation convexe.

**Algorithme 4** Schéma de Transformation Locale : *Descente*

1: générer une solution initiale (e.g. par un algorithme glouton) ;
2: **Tant que** un optimum local n'est pas atteint **Faire**
3: générer des solutions voisines ;
4: sélectionner une solution améliorante dans le voisinage ;
5: remplacer la solution courante ;
6: **Fin tant que**

Introduit par [Kirkpatrick et al. (1983)], le **Recuit Simulé** - ang. *Simulated Annealing* -, autre célèbre *Recherche Locale*, maintient l'objet courant dans les contraintes, et se différencie par le fait d'accepter des transformations dont l'impact sur la performance de l'objet courant est négatif, selon une probabilité indexée sur une variable de température qui diminuera au fil des itérations (et le processus acceptant donc de moins en moins des transformations dégradantes).

La **recherche Tabou** maintient également la réalisabilité de l'objet courant et autorise elle aussi les transformations dégradantes : en sauvegardant, au fil des exécutions, la liste des derniers mouvements effectués et des paramètres associés ([Glover (1990)]). Cette liste, appelée liste Tabou, permet d'éviter d'éventuels cycles dans l'exécution d'un processus de résolution déterministe : à chaque itération, l'opérateur $T$ est appliqué avec une valeur de paramètre $u$ qui n'est pas dans la liste Tabou et dont l'impact sur la performance est le meilleur possible. La taille de la liste est un paramètre important du processus, dont la valeur oscille le plus souvent entre 5 et 20.

**Cas de la méthode GRASP.** La méthode GRASP - ang. *Greedy Randomized Adaptative Procedure* - a été introduite par [Feo and Resende (1989)]. Cette méthode itérative est présentée comme hybride puisqu'elle combine à la fois la recherche gloutonne *randomisée* et une Transformation Locale ou plus précisément une Descente. Le premier composant, l'heuristique gloutonne, construit un objet réalisable $z$, à partir duquel une boucle aléatoire de Recherche Locale (en général une Descente) est appliquée. L'implémentation non déterministe (*Randomization*) du procédé d'initialisation gloutonne permet l'exécution d'un nombre potentiellement important de réplications de l'enchaînement, à l'issue de quoi, la meilleure solution obtenue est conservée.

**Les schémas autorisant une violation de contraintes**

On mentionnera dans ce paragraphe surtout les schémas de pénalité et de décomposition lagrangienne.

Le **schéma de pénalité** est le plus simple et le plus intuitif. Il sépare tout d'abord les contraintes CONT en deux familles : CONT = CONT1 ET CONT2. CONT2 représente les contraintes les plus difficiles (dans le sens où elle freine la recherche de solutions réalisables). Ensuite, la fonction $fcont2(z)$ est créée afin de mesurer le degré de violation des contraintes de CONT2 par l'objet courant $z$. La valeur $fcont2(z)$ est croissante selon ce degré de violation et nulle si CONT2 est satisfaite. Les contraintes CONT2 étant relâchées, on obtient alors, z ne respectant que CONT1, la fonction Objectif de départ pénalisée d'un composant $p.fcont2(z)$, $p > 0$. Plus le processus avance plus $p$ devient important afin de contraindre les différents objets $z$ de respecter *de plus en plus* les contraintes de CONT2.

La **décomposition lagrangienne** est un second procédé plus difficile à mettre en place que le schéma de pénalité mais elle peut s'avérer très efficace. Elle suit le même principe dans le sens où la fonction Objectif est complétée d'expressions basées sur les éléments des contraintes relâchées. Le principe de départ est le même : une partie des contraintes est considérée comme difficile (CONT2) et la recherche de solutions devient bien plus facile une fois les contraintes de CONT2 relaxées. Ces contraintes sont ensuite réinjectées dans la fonction Objectif et pondérées par les **multiplicateurs de Lagrange**, dont le rôle est de permettre une concaténation des conditions d'optimalité locale. Il existe des méthodes intégrant à la fois un schéma de Pénalité classique et de la décomposition Lagrangienne, on parle alors de **lagrangien augmenté**.

### 1.3.3 Exploration Arborescente

Les algorithmes à base d'Exploration Arborescente permettent d'énumérer partiellement ou complètement l'ensemble des solutions. On retrouve, dans la conception de ces algorithmes, les principaux composants des algorithmes gloutons :

— l'espace $S'$ dans lequel se développe l'objet $z$. $S'$ contient un objet vide, un objet noté $Fail$ et l'espace $S$ des objets de même type que $z$ ;
— l'opérateur de construction $C$ qui agit sur les objets $z$ de $S'$. Si $z$ n'est pas complet, $C$ est spécifié par un paramètre $u$ afin de restreindre le domaine de liberté de $z$ en y ajoutant une information non triviale ;
— le paramètre $u$ comprend 2 composants $(v, w)$ : $v$ identifie la nature de la contrainte rajoutée (par adresse) et $w$ sa valeur. Ainsi, si $z$ est un vecteur booléen, $v$ désignera un indice du vecteur et $w$ balayera les deux valeurs possibles $z(u) = 0$ et $z(u) = 1$.

L'arbre d'exploration induit est alors défini de façon implicite : à partir d'un nœud (objet partiel) courant, on fixe $v$ et on obtient des fils de ce nœud pour chaque valeur de $w$ possible. On s'aperçoit rapidement que l'exploration complète des solutions prend rapidement trop de temps. Afin de le rendre plus court, on procède à un **filtrage**. Ce filtrage, de différents types, devra alors permettre de minimiser la taille de l'arbre à explorer.

Le procédé de **filtrage empirique** est le plus simple à décrire. Il va imposer une borne représentant le nombre maximum de nœuds. On se rapproche ici des algorithmes gloutons puisque le nombre maximum de nœuds va impliquer la recherche des meilleurs paramètres de l'opérateur $C$.

Une première catégorie de filtrages non triviaux sont les procédés de **filtrage par anticipation de la qualité**. Le plus connu est celui de Séparation et Evaluation - ang. *Branch and Bound* -. Il met en jeu un procédé de calcul, simple, qui fournit, pour chaque objet partiel $z \in S'$, une borne supérieure $B(z)$ de la qualité optimale des solutions compatibles avec $z$. On renonce à chercher à partir de $z$ si cette borne s'avère moins bonne qu'une valeur qu'on sait être en mesure d'assurer. Si le calcul de $B(z)$ fait apparaître une solution réalisable dont la valeur est égale à $B(z)$, alors on cesse aussi d'explorer à partir de $z$ (mécanisme de stérilisation). D'autres procédés de filtrage reprennent le même principe comme l'algorithme A* (souvent utilisé en Intelligence Artificielle).

Un autre type de filtrage essentiel pour l'exploration arborescente est le **filtrage logique** (ou par **propagation de contraintes**). Il s'agit de réaliser des déductions sur l'objet $z$ courant afin de guider l'exploration de l'arbre. Ceci permet alors d'atteindre plus rapidement une solution cohérente, de réduire le domaine des variables considérées ou encore de montrer qu'un problème n'a pas de solution. Contrairement à ce qui se passait pour les filtrages par évaluations, qui amènent à supprimer la partie de cet arbre qui est en dessous d'un nœud courant, le filtrage logique fait que l'on se déplace dans cet arbre en effectuant des sauts en profondeur. On peut rendre intuitif le fonctionnement d'un filtrage par propagation de contraintes, en considérant l'information définissant un objet partiel $z$ de $S'$ comme une famille de faits, venant se greffer sur une famille de faits de base définissant le problème, de façon à former une base de faits $B(z)$. Propager l'information ayant permis de définir l'objet $z$ signifie alors appliquer de façon chaînée (Chaînage Avant), un certain nombre de règles, une règle se définissant comme un couple $(Premisse, Consequent)$, $Premisse$ et $Consequent$ s'écrivant :

- $Premisse = (Fait_1(var)..Fait_n(var))$,
- $Consequent = (Consequent.Fait, Consequent.Action)$ avec :
  - $Consequent.Fait = (Fait'_1(var)..Fait'_p(var))$
  - $Consequent.Action = (Action_1(var)..Action_k(var))$, où $var$ désigne un ensemble de variables.

La règle devient applicable s'il est possible d'unifier $Fait_i(var), i = 1..n$ avec $B(z)$, c'est-à-dire s'il est possible de trouver des valeurs $val$ pour les variables de telle sorte qu'après substitution de $var$ par $val$, chaque $Fait_i(val)$ puisse être considéré comme étant dans $B$. Dans ce cas, on insère aussi dans $B$ tous les faits $Fait'_j(val)$, en prenant soin d'éliminer les redondances et on exécute les actions $Action_1(var)..Action_k(var)$. Le couple $(var, val)$ s'appelle alors un unificateur. Le schéma global est donné par l'algorithme 3.5.

**Algorithme 5** Propagation de contraintes

**Entrées:** un objet non complet $z$ et une liste de faits $FRacine$ ;
**Sorties:** un booléen $Succes$, $LS$ et $LS1$ ;

$LF \leftarrow FRacine$ ; $Succes \leftarrow VRAI$ ; $LS \leftarrow NIL$ ; $LS1 \leftarrow NIL$ ;
**Tant que** $LF \neq Nil$ ET $Succes$ **Faire**
  $F \leftarrow Tete(LF)$ ; $LF \leftarrow Defiler(LF)$ ;
  **Pour tout** $R$ dans l'ensemble $REGLE$ des règles **Faire**
    $U \leftarrow unifier(Premisse(R), B(z))$ ;
    **Si** $U$ peut s'écrire $(Var, Val)$ **Alors**
      $NouveauxFaits \leftarrow Instancier(Consequent.Fait(R), Var, Val)$ ;
      $Actions \leftarrow Instancier(Consequent.Actions(R), Var, Val)$ ;
      **Si** $NouveauxFaits$ contient des ratés **Alors**
        $Succes \leftarrow FAUX$ ;
      **Sinon**
        **Pour tout** $f$ de $NouveauxFaits$ **Faire**
          **Si** $f$ n'est pas dans $B(z)$ **Alors**
            insérer $f$ dans $B(z)$ et dans $LS$ ;
          **Fin si**
        **Fin pour**
        **Pour tout** $a$ dans $Actions$ **Faire**
          appliquer $a$ à $z$ et $B(z)$ ;
        **Fin pour**
        sauvegarder dans $LS1$ l'ancien contenu des cases mémoires modifiées ;
      **Fin si**
    **Fin si**
  **Fin pour**
**Fin tant que**
**Retourner** $Succes$, $LS$, $LS1$ ;

L'utilité des listes de sauvegarde $LS1$ figurant dans le résultat de l'algorithme 5 tient au fait que si cette fonction est exécutée dans le cadre d'un parcours d'arbre en profondeur (backtracking), il faudra être en mesure de restituer, à chaque retour arrière, l'objet partiel $z$ tel qu'il était avant la propagation.

# Chapitre 2

# Les systèmes innovants de mobilité

## 2.1 Les services innovants de transports

En 2008, et pour la première fois dans l'histoire de l'humanité, plus de la moitié de la population mondiale vit en zone urbaine. Les transports de masse sont de plus en plus nécessaires au bon fonctionnement d'une ville alors que la majorité des personnes continuent à utiliser leur propre véhicule, ce qui est à l'origine d'un grand nombre des problèmes connus dans les centres villes (lire [Dupuy (1999)], [Castex (2007)] et [Chevrier (2008)]). [Valejo et al. (2004)] évalue l'utilisation des différents types de transports au sein de la ville de Coimbra (Portugal). Il s'avère que 66% des usagers utilisent un véhicule individuel, taux bien supérieur à celui de l'usage des transports publics qui arrivent en deuxième position avec seulement 14% (cf. le détail en figure 2.1). (A titre de comparaison, les américains utilisent pour 88% d'entre eux un moyen de transport individuel pour lequel, dans la plupart des cas, le conducteur est l'unique passager [Folsom (2011)]).

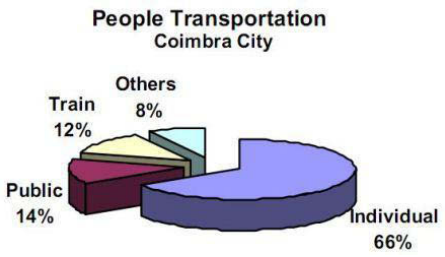


FIGURE 2.1 – Taux d'utilisation des transports dans la ville de Coimbra

Coïmbra reflète ce qui se passe dans les différentes villes européennes, et l'idée fait son chemin que de nouvelles solutions présentant à la fois des caractéristiques des transports publics et individuels pourraient attirer un grand nombre de personnes. [Dupuy (1995)] explique que l'automobile est "associée à un sentiment de liberté, de choix, de libération, s'opposant à l'obligation, à la coercition, à la contrainte [...]. De là naît un sentiment d'autonomie et de pouvoir sur le temps, absent dans le transport en commun [...]. Le système automobile signifie donc depuis toujours libération, puissance, maîtrise du temps et de l'espace". Mais l'image de la voiture traditionnelle

s'est ternie. Elle est maintenant aussi synonyme de pollution, de congestion dans les centres villes, de nuisance sonore et de coût important [Chevrier (2008)]. Dans cette section, mais également dans la suivante, nous présentons les différents systèmes innovants de mobilité dont l'émergence fait suite au constat ci-dessus.

### 2.1.1 Le transport à la demande

Les transports publics ne sont pas suffisamment flexibles dans l'espace et dans le temps. Pour rapprocher logique de mutualisation et impératif de flexibilité, les transports à la demande (TAD) - ang. *Transport On-Demand* ou *On-Demand Transportation* - émergent dans les années 70, notamment à la suite de la politique américaine de l'époque qui met en place, à travers le pays, plus de dix importants projets concernant les transports du futur. Mais ces projets ciblent en grande partie le Fret. Il faut attendre les années 90 pour que la question des TAD soit posée pour le transport de personnes. Les nouvelles technologies de communications en sont le principal facteur.

Les TAD permettent aux usagers de définir un cadre spatio-temporel dans lequel leur déplacement doit s'intégrer. Plus précisément, ils définissent la zone de ramassage et la période pendant laquelle ils veulent être pris en charge ; le même type d'information est également fourni pour la destination. Selon les services, les périodes et les zones sont plus ou moins précises. Du fait de la relativement restreinte taille du véhicule (de la voiture au minibus), plusieurs auteurs considèrent le TAD comme un système de taxis partagés. Les trajets sont rarement directs, les exploitants essayant de remplir leur véhicule. Nous nous baserons sur la définition de [Ascher (2002)] qui inclut les flottes de taxis classiques en définissant le terme "transport à la demande" comme "une notion générique qui englobe a priori tous les services de transport dont tout ou partie ne s'effectue qu'à la demande expresse de ceux qui les utilisent".

Si l'on compare aux Etats-Unis et aux pays voisins, les transports à la demande tardent à se développer en France mais le nombre de système de TAD y est tout de même relativement important (plusieurs centaines). La plupart restent aujourd'hui réservés aux personnes à mobilité réduite ou aux malades. Il existe tout de même d'autres types d'exploitations. Il faut différencier les TAD du milieu rural et ceux du milieu urbain. Dans le premier cas, il est souvent mis en place pour désenclaver et assurer le droit à la mobilité pour tous. Dans les villes (grandes mais aussi moyennes), les TAD jouent un rôle non négligeable en cas de déficience des systèmes de transport en commun. Clermont-Ferrand propose par exemple un premier service TAD de personnes à mobilité réduite et un second, pour tout public, desservant les villes voisines ([Moovicite (2013)]).

Les grandes sociétés françaises commencent à s'intéresser au transport à la demande : il en est ainsi pour VEOLIA qui a d'ailleurs récemment racheté *SuperShuffle*, un société pionnière aux Etats-Unis dans le TAD d'aéroports. Cette société qui propose aujourd'hui ce service en France ([SuperShuffle (2013)]) a un chiffre d'affaires global qui s'élève à plusieurs dizaines de millions de dollars.

Les pouvoirs publics poussent au développement de ce type de transport. L'agence nationale de la recherche (ANR) a financé le projet Modulobus de 2008 à 2011. Celui-ci consistait à étudier un nouveau système de transport à la demande s'appuyant sur un prototype de logiciel basé sur "des systèmes informatiques centralisés (optimisation globale) et distribués (cartographies, feuilles de routes et informations clients), lui permettant d'optimiser ses tournées en temps quasi-réel, en s'autorisant des détours raisonnés et en tenant compte de l'organisation spatiale de la flotte de véhicules" ([Modulobus (2008)]). Plusieurs expérimentations ont suivi comme le *Modulobus de Noël* qui a permis aux habitants de Montbéliard d'être transportés jusqu'au marché de Noël à partir de l'ensemble des 529 arrêts de bus de la ville. Les usagers pouvaient réserver le dimanche pour un transport dans l'heure [MdN (2003)]. Le module d'optimisation du logiciel de gestion de la flotte a permis une réponse très rapide pour les usagers, à l'image de ce qui se fait à l'étranger (Flexline, Shuttle etc.). Cette réactivité est rare (notamment en France), la plupart des exploitations d'aujourd'hui n'intègrent encore que trop rarement une gestion dynamique "pure" des demandes et la réponse aux requêtes (acceptation de la demande et communication d'une date de rendez-vous) est très rarement quasi instantanée. Les problèmes de Recherche Opérationnelle qui en découlent sont compliqués et cette thèse tente d'y apporter un premier élément de réponse à visée opérationnelle.

### 2.1.2 Les véhicules partagés

Parmi les systèmes de transports récents, ceux qui attirent majoritairement les usagers sont, et ce n'est pas étonnant, les plus économiques. Le partage de véhicules en fait partie et il a changé la manière de voyager pour beaucoup d'entre nous. Son faible coût est dû au "rendement" du partage des véhicules, et ceci à deux niveaux.

Le premier niveau de partage consiste à disposer d'un véhicule pendant une durée établie avant de le ramener dans une station prévue à cet effet. Plusieurs exploitations privées de ce type de transports existent aujourd'hui. En Allemagne, plus précisément dans la ville de Brême, un premier service de véhicules partagés est en place [Bremen (2005)] ; à Québec, Sherbrokke, Ottawa, Gatineau et Montréal, les canadiens peuvent utiliser ce type de service, proposé par Communauto [Communauto (2013)] ; les Etats-Unis ne sont pas en reste : ainsi la société *IGO Car Sharing* propose un grand nombre de véhicules pour circuler dans la ville de Chicago [Igocars (2013)] et cela sans aucun but lucratif ; les lillois ont la possibilité de conduire des voitures partagées proposées par la société *Lilas* [LILE (2007)] à l'instar des Parisiens qui connaissent depuis 2011 un tel service mais avec des voitures éléctriques : *autolib* [Autolib (2011)]. Ce service en plein essor compte déjà plus de 1000 véhicules dans la capitale et a commencé à s'exporter à l'étranger un an après sa mise en service. Ce type de partage tend à faire disparaître les locations de véhicules classiques (pour les trajets intra-muros) d'autant plus que plusieurs catégories de véhicules sont représentées comme les deux roues non motorisés (comme les *Vélib* [Velib (2007)] dont le concept a d'ailleurs été exporté en Chine, *CVélo* à Clermont-Ferrand, etc.) ou encore les utilitaires. Au passage, ces sociétés ont le même problème majeur d'optimisation

combinatoire : celui du rééquilibrage des stations - ang. *balancing the stations* -. En effet, les origines ou les destinations des usagers, sur un trajet donné, se rejoignent, créant ainsi un déséquilibre de population de véhicules disponibles dans les stations. Aussi, certains usagers n'utilisent ces véhicules que pour des portions de chemins difficiles (montée, chargement lourd etc.) et ne font donc pas de trajet retour. Il est probable que la résolution de ce problème trouve sa place dans les colonnes de la littérature spécialisée de la Recherche Opérationnelle, les années à venir.

Le second niveau élargit la mutualisation du véhicule : ici, plusieurs usagers ou groupes d'usagers, indépendants les uns les autres, peuvent être amenés à partager la capacité du véhicule. Nous parlons ici des transports en commun mais surtout du covoiturage. En 2012, nous fêtions le $350^{\text{ème}}$ anniversaire de la mort du clermontois Blaise Pascal. Ce dernier a posé une des premières pierres du transport public urbain avec son fameux carrosse à cinq sols. L'existence même des transports publics a fait naître l'idée que des individus pouvaient voyager dans un même véhicule sans même se connaître. Le covoiturage, très en vogue aujourd'hui et au coût très avantageux pour les particuliers, relève également de cette idée. Il concerne différents types de transports, mais surtout les trajets domicile-travail ([CERTU (2007)] et [ADEME (2010)]). Cependant, les longs trajets, comme lors des départs en vacances, sont de plus en plus courants et ceci grâce à l'essor des moyens de communication. Sur un site internet spécialisé dans le covoiturage, le propriétaire du véhicule propose un trajet laissant une certaine flexibilité en terme de temps (temps maximum de trajet) et d'espace (détours). Les usagers clients vont alors établir un lien directement sur la toile par ce site en formulant leurs propres contraintes. Le site le plus utilisé en France, compte plus de 2 millions de membres ([Covoiturage.fr (2006)]) ; bien que tous ne se sentent pas encore prêts à voyager avec des inconnus et ne se sont inscrits que par curiosité, des centaines de milliers de voyages ont été réalisés. Ce très grand nombre concerne en majorité des jeunes, ce qui permet d'imaginer pour ce type de transports un succès encore plus grand à l'avenir. De plus, ces sites s'accommoderont à merveille des réseaux sociaux où les groupes "d'amis" prennent connaissance des différents voyages réalisés par leurs "proches". Notons au passage que bien des personnes ne rechignent pas à se retrouver dans la même voiture qu'un ou plusieurs inconnus. Le caractère dynamique du covoiturage est également envisagé par le CERTU (Centre d'Etudes sur les Réseaux, les Transports, l'Urbanisme et les constructions publiques). L'idée du covoiturage dynamique est "de fournir une offre en quasi temps réel à l'usager covoitureur". La personne souhaitant effectuer un itinéraire en covoiturage contacte le service quelques minutes avant son départ. Le service va alors chercher le conducteur adéquat qui est en mesure d'offrir le covoiturage souhaité sur l'itinéraire demandé même si le véhicule est déjà en train d'effectuer un trajet. Le centre d'étude devrait rendre ses conclusions dans un avenir proche [Centre d'Etudes sur les Réseaux (2009)]. Il fonctionne déjà aux Etats-unis grâce au service *Avego* [Avego (2013)] auquel les utilisateurs peuvent envoyer leurs requêtes directement avec leur portable. La construction de l'ordonnancement se fait par la résolution d'un problème proche du DARP dynamique.

Il est déjà envisagé qu'un nouveau type de service allie les caractéristiques de ces deux niveaux de partage avec l'utilisation de véhicules tampons. Ici, le véhicule est également prêté au conducteur qui est choisi par les passagers. L'attribution du véhicule et le traçage des tournées se font en amont (demandes gérées de manière statique) comme pour le covoiturage classique. Nous pouvons imaginer - et même jusqu'à une très grande échelle (les véhicules étant facilement traçables par GPS que ce soit pour les usagers ou l'exploitant) - le parcours d'un de ces véhicules dont le volant passerait entre plusieurs mains, ceci avec un taux de remplissage important. Il faudrait cependant que chaque individu ne perde pas trop de temps à rejoindre sa destination sachant que plusieurs étapes intermédiaires sont à réaliser pour satisfaire les autres usagers : c'est le rôle des contraintes et/ou de l'optimisation. La problématique légale est finalement la même qu'*autolib*. Reste le choix du conducteur. L'optimisation d'une flotte de véhicules tampons se fait par la résolution du *Dial-a-Ride Problem* statique dans lequel on contraint les tournées à n'avoir aucune portion de route sans conducteur. Ce service peut également être envisagé dans un contexte dynamique. Aussi, sur de longs trajets, un transbordement entre deux véhicules tampons est envisageable à l'image des correspondances pour les trajets en train. En ce qui concerne la supervision, c'est le DARP avec préemption de véhicules qui est mis en jeu.

L'émergence de ces services sera également confortée par le système des voitures dites connectées qui permettra de connaître (par l'intermédiaire de web services) l'état du réseau routier ainsi que la position des véhicules (si l'échange de données entre constructeurs est effectif). L'évolution technologique des véhicules implique de nouvelles problématiques dont l'une, importante, relative au caractère "sans conducteur", est traitée dans cette thèse.

## 2.2 Les véhicules de nouvelles générations

### 2.2.1 Besoins et cibles d'un nouveau type de véhicule

Parmi les problématiques actuelles de mobilité, la question récurrente des embouteillages apparaît fréquemment suite à l'engorgement des carrefours. Aujourd'hui plusieurs chercheurs envisagent une plus grande fluidité de la circulation au croisement des routes grâce à une gestion automatique des véhicules comme [Dresner and Stone (2008)] développant un système multi-agents montrant qu'en tirant parti des capacités des véhicules autonomes, il était possible d'accélérer le trafic de façon spectaculaire.

La sécurité des transports est également un sujet important. Dans le monde, les accidents de la route sont la cause de 1,2 million de morts (3250 en France pour l'année 2013) et 50 millions de blessés chaque année [Routière (2008)]. Les facteurs humains sont en cause dans plus de 90% des accidents corporels [PrRoutiere (2012)] comme le montre le tableau 2.1.

| Facteurs humains dans les accidents | Taux de décès et d'accidents |
|---|---|
| Conduite sous influence de l'alcool | 28.5% des tués |
| Vitesse excessive | 18.5% des tués |
| Défaut de ceinture | 23% des tués |
| Téléphone au volant | $>$ 7% des accidents corporels |
| Présence d'un obstacle fixe | 35% des tués |
| Fatigue et somnolence au volant | 30% des accidents mortels sur autoroute |
| Conduite sous influence du cannabis | $>$ 3% des tués |
| Conduite sans permis ou sans assurance | 4% des tués |

TABLE 2.1 – Les principaux facteurs humains dans les accidents de la route en France

Partant de ce constat, le Japon, les États-Unis et l'Europe ont étudié les ITS (Intelligent Transport System) en développant, chacun, deux importants projets :

- Japon :
  - Advanced Cruise-Assist Highway Systems [AHSRA (1989)],
  - Advanced Safety Vehicle [Sato (1991)] ;
- État-Unis :
  - Automated Highway Systems, [Schneider (1990)],
  - Intelligent Vehicle Initiative, [IVI (1998)] ;
- Europe :
  - Eureka Prometheus, [Prometheus (1986)],
  - Advanced Driver Assistance Systems in Europe, [ADASE (2000)].

Ces projets ont comme objectif commun l'étude des nouvelles technologies en informatique, automatique, télécommunications et électronique afin d'améliorer l'efficacité et la sécurité du transport terrestre. Mais ici, encore une fois, le véhicule reste privé avec tous les problèmes d'espace que cela implique (encombrement, temps de trajet en augmentation, manque de places de parking, pollution, nuisances pour les citadins etc.) [Fraichard (2005)]. Ces constatations sont sans appel : un nouveau type de transports est nécessaire et un de ceux étudiés dans le cadre de cette thèse, basé sur l'utilisation de véhicules électriques autonomes de petite taille, semble être une bonne solution pour l'amélioration de la sécurité et de la qualité de vie des citoyens. Il est d'ailleurs à noter que ce service peut également être exploité dans bien d'autres contextes :

- complexes médicaux,
- campus,
- centres d'affaires,
- grands parking,
- aéroports et gares
- grandes manufactures.

Chacune de ces utilisations implique une complexité variable dans la gestion de la flotte de véhicules automatisés en partie due à différentes contraintes de fiabilité. Les PRT et cybercars sont nés de l'idée de résoudre les problèmes précédemment cités.

### 2.2.2 Les systèmes et véhicules autonomes

**Le Personal Rapid Transit**

Le concept des PRT datant des années 60 a été formalisé en 1988 par [ATA] en 7 points :

1. véhicules entièrement automatisés sans conducteur humain,
2. véhicules qui utilisent une voie la plus étroite possible et qui leur est dédiée,
3. véhicules de taille réduite au minimum et disponibles 24h/24h pour une utilisation exclusive pour un groupe de personnes (par exemple de 1 à 6 passagers réunis selon leur propre choix),
4. voies étroites dédiées, au niveau du sol, au-dessous ou au-dessus,
5. véhicules qui doivent pouvoir atteindre chaque station et utiliser chaque voie du réseau,
6. parcours direct entre un point de départ et une destination (aucun arrêt aux stations intermédiaires),
7. service disponible à la demande et non selon des horaires fixes.

Aujourd'hui, l'émergence des PRT est ralentie par les coûts de l'infrastructure. De plus, l'utilisation des voies dédiées, même avec leur étroitesse, limite leur utilisation en milieu urbain. Les parcours directs n'optimisent pas non plus l'utilisation de la capacité des véhicules. Le nombre d'exploitations de ces services est très faible, on peut citer le service ULTRA [PRT (2011)] de l'aéroport Heathrow de Londres et le tout premier PRT exploité à Morgantown, en Virginie-Occidentale aux États-Unis, en service depuis 1975. Nous allons voir que le concept des PRT est relativement proche des Cybercars. Ces derniers ont la particularité de pouvoir être déployés sur pratiquement n'importe quel espace. Ceci implique évidement de nouvelles contraintes pour la plupart liées à ce que les véhicules pourraient rencontrer sur leur chemin.


FIGURE 2.2 – Les PRT de Heathrow et Morgantown

**Le Cybercar**

Les Cybercars sont des véhicules terrestres offrant une conduite automatique sans aucune intervention humaine. Pour cela, ils sont dotés de plusieurs capteurs qui permettent d'analyser l'environnement et éviter tout obstacle. Une flotte suffisamment importante de ces voitures automatiques est à même de constituer un système de transports qui peut être managé par une entité centrale communicante. Proches des PRT, ils offrent l'avantage d'être capables de rouler sur tout type de terrain [Analysis (2000)]. Contrairement aux PRT et malgré plusieurs expériences "sur le terrain", il n'existe pas encore à ce jour d'exploitation commerciale d'un service de transport de personnes par les Cybercars. Depuis le début des années 2000, les PRT et les Cybercars ont fait l'objet de plusieurs études et d'expérimentations avec le support d'importants projets européens. Les Cybercars ne sont pas exploités commercialement à l'heure actuelle. [Berger et al. (2011)] considère la technologie moins avancée que celle des PRT. Cependant, leur praticité explique l'engouement de plusieurs laboratoires de recherche pour le sujet (Institut Pascal (ex LASMEA) et INRIA entre autres). Ils résument en quatre points leurs principaux avantages tirés des projets Cybercars, CyberMove et CytiMobil ([Cybercars], [CyberMove] et [CityMobil]) :

— disponibilité : les usagers peuvent trouver des véhicules n'importe quand dans la journée,
— facilité : les personnes à mobilité réduite peuvent emprunter ce type de transports,
— redéploiement : Les Cybercars se déplacent facilement d'un endroit à un autre et peuvent utiliser les aires de stationnement laissant ainsi libre l'espace urbain (contrairement par exemple aux tramways mais aussi aux PRT en raison de leur infrastructure respective nécessaire),
— réduction de la congestion des villes : amélioration de la qualité de l'air et diminution de la déperdition d'énergie.

**Quelques exemples : Le VIPA et autres Cybercars.** Le véhicule automatisé *Cycab* (figure 2.3) de l'Institut Pascal perçoit l'environnement par capteurs GPS, GPS différentiels centimétriques, télémètres LASER, caméras et odométrie. La vitesse maximum d'évolution est de 18km/h. Des démonstrations grandeur nature de guidage par vision ont déjà été réalisées dans les centres villes de Clermont-Ferrand (place de Jaude) et Nancy.

FIGURE 2.3 – Deux Cycabs de l'Institut Pascal (ex LASMEA)

Il existe bien d'autres Cybercars comme le *Yamaha AGV Cybercar*, les *Serpentines Cybercars* (figure 2.4) qui ont fait l'objet d'une démonstration grandeur nature à Lausanne, ou encore le *Cybus* et le VIPA.

FIGURE 2.4 – Le Yamaha AGV à Coimbra et les Serpentines à Lausanne

Le Véhicule Individuel Public Automatique ou VIPA est un Cybercar réalisé grâce au partenariat entre un laboratoire de recherche, un bureau d'études et un constructeur automobile respectivement :

— l'Institut Pascal (ex LASMEA) [1] qui a travaillé sur le système de navigation,
— APOJEE qui a fourni son expérience en électronique embarquée,
— LIGIER [2] qui a dessiné et produit le véhicule.

Le VIPA est électrique, peut transporter jusqu'à 6 personnes (dont 4 assises) à une vitesse de 5 à 20km/h et pèse environ 1 tonne. Il dispose d'un système de localisation et de guidage basé sur le traitement d'images et d'algorithmes en temps réel. Il ne nécessite aucun chauffeur et aucune infrastructure de guidage (plots, lignes au sol etc.). Que ce soit individuellement ou en flotte, le VIPA peut fonctionner en présence de piétons et éventuellement de véhicules à faible vitesse.


FIGURE 2.5 – Le VIPA au salon de l'automobile de 2010

Grâce à sa mémoire visuelle il réalise un apprentissage de la trajectoire ce qui lui permet d'être déployé facilement. Par exemple, pour mettre en place une flotte de VIPA le long d'un circuit, un chauffeur conduit la voiture de tête durant un

1. http://http://ip.univ-bpclermont.fr/r/
2. http://www.automobiles-ligier.com/

seul tour alors que les autres véhicules le suivent de manière automatique. Le tour effectué, le VIPA de tête a enregistré toutes les trajectoires, s'est représenté en 3D les caractéristiques du parcours et toute la flotte peut alors se passer d'aide humaine.

Puisque le VIPA n'a pas besoin d'infrastructure particulière le coût d'exploitation est considérablement réduit d'autant plus que la technologie utilisée n'est pas très onéreuse. Un véhicule ne coûte que 50000€ alors qu'il n'est pas encore intégré à une production de masse, les *Cybus* coûtant 150000€ et le matériel des voitures automatiques de *Google* 150000$.

#### Les Grands Projets

L'union européenne a initié plusieurs projets dans le domaine des transports urbains. CyberCars, CyberMove et CityMobil sont les premiers projets qui se sont intéressés de près à la technologie des véhicules individuels autonomes.

Les projets Cybercars et CyberMove (2001-2005) furent menés en partie par l'INRIA et créés par un partenariat entre 7 laboratoires de recherche et autant de compagnies industrielles. Le premier, plus technique, a permis l'échange des meilleures pratiques dans le domaine des Cybercars, avec l'évaluation des véhicules, du management de flottes de Cybercars, de la consommation électrique et de la fiabilité. Le second projet, CyberMove, avait pour objectif de mettre en avant l'utilisation d'un tel service et de démontrer l'apport de celui-ci dans plusieurs villes européennes.

Le programme européen [CityMobil (2011)] financé à hauteur de 40 millions d'euros, aura pendant plus de 5 années (de Mai 2006 à Octobre 2011) étudié une nouvelle organisation des transports à l'intérieur des grandes villes afin de limiter les embouteillages et la pollution mais aussi de proposer des transports plus sécurisés et une meilleure intégration de ceux-ci dans l'espace urbain. Pour cela, le projet a développé de nouveaux outils pour le management des transports et s'est penché sur le développement de nouveaux types de véhicules.

D'autres projets existent comme celui du géant Google qui a dévoilé ses premiers travaux sur six Toyota Prius et une Audi TT auxquelles ont été adaptés un nombre important de capteurs sensoriels et de caméras vidéos. Ces véhicules ont déjà parcouru des milliers de kilomètres sans aucune aide humaine et plusieurs centaines de milliers simplement avec une aide occasionnelle [Google (2010)], mais la firme n'en dévoile pas plus pour l'instant sur les conditions de ces expériences entièrement automatisées.

### 2.2.3 L'intégration des véhicules autonomes

Petit à petit, le statut légal des véhicules sans conducteur progresse. Mi-2013, 3 états (Floride, Nevada et Californie) les autorisent : des études sont déjà en cours au niveau fédéral. En France comme en Europe, ces véhicules ne sont toujours pas acceptés sur la voie publique ([Project (2000)]). Les Cybercars feront probablement leurs preuves et seront certainement intégrés pleinement dans la circulation. En 2011 et pendant 2 mois, la deuxième démonstration du projet *CityMobil* et son véhicule *Cybus* (cf. figure 2.6) permettait au public de les utiliser. Les véhicules évoluaient sur circuit dédié avec 5 stations, seules quelques intersections leur faisaient rencontrer d'autres véhicules mais ils y gardaient la priorité. La figure 2.6 schématise le parcours réalisé par ces véhicules. Les véhicules qui ne sont pas du côté des stations font un crochet pour les atteindre. Nous verrons dans le chapitre 7 comment limiter ce court-circuitage.

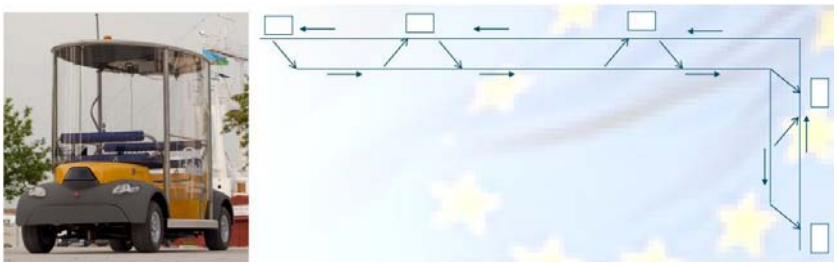

FIGURE 2.6 – Mise en service du *Cybus* à la Rochelle

En Europe, de plus en plus d'expériences sont réalisées sur le terrain par ces véhicules intégrés à un nouveau type de transports. Ils devraient être le pont entre les services existants, ils permettent des parcours multimodaux plus longs limitant ainsi la marche à pied (mais nous voilà sur un tout autre débat...). En mars 2013, d'autres expériences de ce type ont été réalisées à Lyon par un véhicule proche du *Cybus* appelé *Navia* et avec une capacité de 8 passagers. La mairie prévoit son exploitation pour relier au tramway les constructions réalisées sur les rives de la Saône, mais toujours sur des voies sans interactions avec d'autres véhicules motorisés. Le VIPA aura quant à lui la possibilité de faire ses preuves au sein du CHU *Estaing* ainsi que sur le site Ladoux de *Michelin*, permettant aux employés de rejoindre les différentes entités du site.

Enfin, ces véhicules, qui ont vocation à être utilisés en flottes pour former un service de transports à la demande, se doivent d'être performants aussi en ce qui concerne leur management. Les problèmes de tournées de véhicules intégrant des contraintes liées à l'autonomie sont nouveaux dans le domaine de la recherche opérationnelle.

## 2.3 Modélisations des problèmes de tournées de véhicules

Cette section a pour objet d'effectuer un bref rappel sur les principaux modèles de transports, en mettant l'accent sur les systèmes innovants présentés dans les sections précédentes. Nous étudions en particulier le transport à la demande modélisé par le DARP.

#### Le problème du voyageur de commerce et le VRP

Les premiers travaux portant sur les problèmes d'optimisation de transports sont associés au fameux problème du voyageur de commerce - ang. *traveling salesman problem* (TSP) - qui remonte à un ou deux siècles. Son origine n'est pas clairement établie bien qu'elle semble se situer au début des années 1800. Le TSP peut s'énoncer simplement comme suit : un voyageur de commerce doit se rendre une seule et unique fois dans un ensemble de villes donné tout en minimisant la distance totale parcourue. Cela intègre une contrainte de cycle (départ et arrivée au dépôt) et une contrainte d'unicité du service (un seul passage par nœud).

Le problème a été formulé à plusieurs reprises par des mathématiciens américains, irlandais (dont l'illustre William Rowan Hamilton qui en créa un jeu : *the Hamilton's Icosian game*) et autrichiens dont Karl Menger à qui on attribue les premiers travaux de résolution [Menger (1932)]. Les TSP entrent dans l'ensemble des problèmes NP-Difficile. La preuve de cette complexité a été donnée suite à celle d'un problème voisin : la recherche d'un cycle hamiltonien pour lequel Richard M. Karp montra qu'il était NP-Complet [Karp (1972)].

La figure 2.7 représente une solution d'une instance du TSP avec 4 villes. Le véhicule part du dépôt, visite les 4 villes et retourne au dépôt.

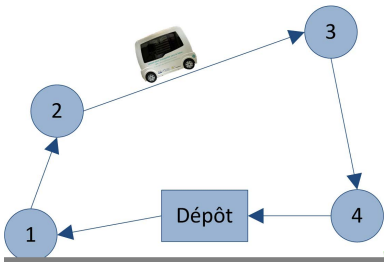


FIGURE 2.7 – Une solution du TSP d'une instance à 4 villes

Le problème de tournées de véhicule - ang. *vehicle routing problem* (VRP) - reprend celui du voyageur de commerce excepté sur ce point que les différents services peuvent être réalisés cette fois par plusieurs véhicules (au nombre de $K$). Certaines modélisations intègrent une contrainte de capacité qui limite chaque véhicule à réaliser un nombre maximum d'opérations. La figure 2.8 montre un VRP avec deux véhicules chacun satisfaisant 4 demandes partant et arrivant d'un même dépôt.

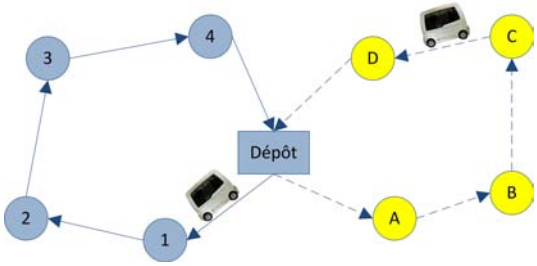


FIGURE 2.8 – Une solution du VRP d'une instance à 8 demandes

**Le *pickup and delivery problem.***

De multiples variantes sont issues du TSP mais la construction du planning des véhicules acheminant des marchandises ou des personnes, le problème de ramassage et de livraison - ang. *Pick Up and Delivery Problem (PDP)* -, peut être considéré comme le principal ancêtre du DARP. Il a été enrichi et de ce fait complexifié avec le temps. Posons-en, en quelques lignes, les principales caractéristiques. Le PDP est un VRP auquel on ajoute des contraintes de couplage sur les nœuds correspondant aux origine et destination d'une même demande, des contraintes de précédence (le ramassage est réalisé avant la livraison), et des contraintes de capacité.

Un ou des véhicules (au nombre de $K$) partent d'un dépôt pour effectuer ces tâches de ramassage et de livraison avant d'y revenir. Le dépôt fait partie de l'ensemble des nœuds $X$ du graphe $G$ associé au problème, les autres nœuds représentant les origines et destinations des demandes à satisfaire. Chaque paire de nœuds peut être reliée par une arête de $E$. On leur attribue généralement une valuation positive synonyme de coûts (comme la distance ou la durée pour rejoindre deux nœuds). Par ces arcs, les $K$ véhicules peuvent parcourir l'ensemble des nœuds du graphe et ceci dans le but de satisfaire l'ensemble des demandes de $D$. Le modèle intègre aussi souvent des contraintes de capacité reflétant ici la quantité maximum de chargement qu'un véhicule peut supporter sur tout arc du graphe. Cette capacité, notée $CAP$, est donc comparée à la somme des charges courantes. Le problème consiste à créer les chemins des $K$ véhicules de telle sorte que chaque demande soit satisfaite tout en répondant à l'ensemble des contraintes. Ces chemins, qui sont appelés tournées, doivent être établis de façon à optimiser un critère de performance. Ce dernier peut être de différents ordres. Le critère le plus fréquent concerne les distances parcourues. D'autres contraintes peuvent également intervenir, les contraintes de fenêtres de temps, par exemple.

Celles relatives au couplage et à la précédence se classent en trois catégories. Les plus fréquentes dans les PDP sont soit 1-1, soit 1-M-1, - ang. *one-to-one problem* et *one-to-many-to-one problem* -. Le premier détermine l'unicité des origines et destinations par demande et le second remplace l'un des deux par le dépôt. Enfin, les problèmes *many-to-many* (M-M) caractérisent le fait que chaque nœud (dépôt compris) peut jouer le rôle d'origine ou de destination (point de ramassage ou point de livraison). L'ensemble de la classification des problèmes de *Pickup and Delivery* est

donné dans [Berbeglia et al. (2007)].

La figure 2.9 représente deux tournées formant la solution à une instance du PDP à 4 demandes. Par exemple, la demande d'index 1 est satisfaite par le passage en o1 (nœud origine ou de ramassage) en amont du passage en d1 (nœud destination ou de livraison). Ici, la capacité des véhicules est au moins supérieure aux deux sommes des charges de chacun des deux couples de demandes pris en charge par un véhicule.

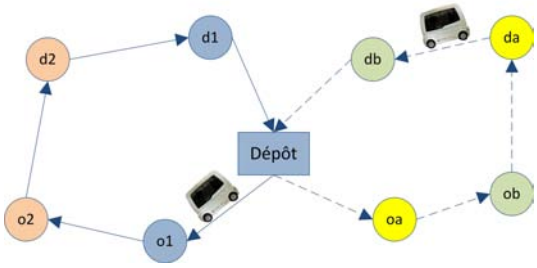


FIGURE 2.9 – Une solution du PDP d'une instance à 4 demandes

**Le *Dial-A-Ride problem***

Le *Dial-a-Ride Problem* est relatif au transport à la demande. Formellement, il n'existe pas de nette distinction entre DARP et PDP. Mais, alors que la sémantique de base du PDP concerne le transport des biens et la logistique, celle du DARP met en jeu des personnes. En conséquence, on met l'accent sur les critères de qualité et les contraintes reflétant les préoccupations d'individus (rendez-vous, temps d'attente, temps de parcours individuels...) ainsi que sur la version dynamique du problème.

Les problèmes de transports de personnes ont fait l'objet de nombreuses études et après une baisse de l'engouement pour le DARP au siècle dernier en raison de problèmes technologiques, il intéresse à nouveau les chercheurs. Cela n'est évidemment pas lié à un phénomène de mode mais à un besoin grandissant d'optimiser les coûts, la qualité de service et même aujourd'hui la robustesse. Mais, à l'avenir, il est probable que ces problèmes seront au cœur des nouveaux services de transports comme nous avons pu le voir en début de ce chapitre. Avant d'établir un état de l'art sur les techniques de résolution, arrêtons nous d'abord sur les caractéristiques élémentaires qui composent le DARP.

**Les demandes.** Ce sont les usagers du service qui forment les requêtes devant être satisfaites par les véhicules. La demande comprend des lieux d'origine et de destination, une information sur la taille de l'objet à transporter (le chargement) et des obligations temporelles à respecter. Ces dernières contraintes consistent en une fenêtre de temps sur chaque point de passage et une durée maximum pour joindre l'origine et la destination.

**Les véhicules.** Une flotte de véhicules, autonome ou pas, est à disposition des usagers pour la satisfaction de leurs requêtes. Si elle dispose de véhicules aux caractéris-

tiques identiques, nous parlons de flotte homogène, sinon de flotte hétérogène. Dans la plupart des cas, la différence s'établit au niveau de leur capacité. Cette dernière peut être modélisée selon différentes mesures comme le poids, le volume ou encore la surface au sol. Des problèmes plus complexes les différencient par des indices de consommation énergétique ou de puissance.

**Les objectifs.** Les critères qu'évalue la performance d'une solution d'une instance du DARP peuvent être nombreux. Ils sont analogues à la fonction Objectif suivant les termes de la programmation linéaire, et l'optimisation peut alors être d'ordre mono-critère ou multi-critères ([Paquette et al. (2007)]). Les quatre critères les plus couramment employés sont :

1. La distance totale parcourue par les véhicules. Ce critère permet par exemple de minimiser la consommation d'énergie ou l'usure des véhicules. Cette distance ne couvre pas les temps des arrêts des véhicules, et n'a donc pas d'effet direct sur les temps de travail, temps d'acheminement, etc... ;
2. Le temps global de l'ensemble des tournées. Cette somme cumule les durées de toutes les tournées et peut être interprétée comme les temps de travail des conducteurs. La minimisation de ce temps semble directement impliquer une "compression" des intervalles entre tous les arrêts des véhicules ;
3. Le temps total pour la réalisation de toutes les connexions. Il est obtenu par l'addition de toutes les durées pendant lesquelles chaque passager est pris en charge par le service, soit le temps entre le chargement et le déchargement de chaque demande. Le terme connexion est ici tiré du vocabulaire des réseaux informatiques, il qualifie la durée de réalisation d'une demande - ang. *ride time* -. Cet objectif peut servir de mesure de la qualité de service ;
4. Le temps d'attente global. C'est la différence entre le temps global de l'ensemble des tournées et la distance totale parcourue par les véhicules. En effet, la très grande majorité des modèles ne sont pas en phase avec la vitesse effective des véhicules ce qui amène à considérer seulement le déplacement du véhicule de manière binaire : soit celui-ci roule à vitesse constante soit il est arrêté. Lorsqu'un véhicule a parcouru la distance séparant deux lieux consécutifs prévus sur son plan de route alors que la prochaine action à réaliser ne peut s'effectuer à l'instant [3], nous parlons de temps d'attente. La minimisation de ces derniers peut être indispensable dès lors que les véhicules sont soumis à la congestion du trafic. Par exemple, pour les véhicules ne pouvant se dépasser (d'autant que certains peuvent être indépendants de la flotte) pour des raisons technologiques (véhicules autonomes) ou de voirie (rues étroites par exemple), il y a tout intérêt à minimiser ces temps pour éviter un embouteillage. Aussi, toute la période passée par les usagers dans les véhicules à l'arrêt est synonyme de désagrément pour eux, surtout si le véhicule est bondé.

3. En raison de contraintes temporelles

Il existe bien d'autres sens de l'optimisation et tout dépend évidement du contexte. Par exemple, si chaque sortie du dépôt implique un coût important, l'exploitant dirigera la supervision de sa flotte vers une optimisation minimisant le nombre de véhicules nécessaires à la satisfaction des demandes. La capacité des véhicules peut également intervenir dans l'objectif à atteindre et cela de deux façons qui peuvent sembler contradictoires : le confort des usagers et le rendement des véhicules.

**Les contraintes.** Les contraintes du PDP sont toutes susceptibles d'être reprises ici. D'autres contraintes, spécifiques au transport de personnes, peuvent être rajoutées. Certaines concernent les temps de connexion (chaque borne est fournie par un usager) et d'autres la durée totale de chaque tournée (donnée par l'exploitant). D'une certaine manière ces bornes poussent l'optimisation dans le sens des objectifs 2 et 3 présentés ci-dessus. Les clients du service fournissent également deux fenêtres de temps, une pour le lieu d'origine et l'autre pour le lieu de destination. Ce sont deux périodes pendant lesquelles le chargement et le déchargement doivent avoir lieu.

L'ensemble de ces contraintes temporelles réduit considérablement le nombre de solutions, ce qui complique le problème d'un point de vue syntaxique, mais permet aussi de filtrer le processus de recherche. Les problèmes de *Dial-a-Ride* étudiés dans cette thèse intègrent l'ensemble de ces contraintes et nous nous intéresserons plus particulièrement aux méthodes rapides et génériques, induisant une continuité entre contextes dynamique et statique. Le tableau 2.2 résume l'ensemble des contraintes des 3 principaux problèmes présentés ci-dessus.

| Contrainte / Problème | TSP | VRP | PDP | DARP |
|---|---|---|---|---|
| Service unique | OUI | OUI | OUI | OUI |
| Cycle (Dépôt) | NON | OUI | OUI | OUI |
| Capacité | | OUI | OUI | OUI |
| Durée Maximum de tournée | | OUI | OUI | OUI |
| Précédence | | | OUI | OUI |
| Couplage | | | OUI | OUI |
| Durée Maximum de connexion | | | | OUI |
| Fenêtre de temps | | | | OUI |

TABLE 2.2 – Contraintes des principaux problèmes de transport

**Les contextes statique et dynamique.** [Psaraftis (1995)] définit les deux contextes par leur différences : le problème des tournées statiques amène à chercher un objet-solution définitif à partir de requêtes connues en amont de la résolution (nous parlons également alors de demandes déterminées), au contraire du problème dynamique qui doit prévoir l'instauration de nouveaux besoins de la part des usagers, quitte à réorganiser ce qui a déjà été établi dans la conception précédente des plans de route. Autrement dit l'objet-solution prévoit en quelque sorte l'arrivée de nouvelles requêtes.

**La préemption.** La plupart des modèles traités dans la littérature sont non *préemptifs*, c'est-à-dire qu'ils impliquent la prise en charge d'une demande en un unique parcours et sans escale. Nous étudierons ici deux variantes de la notion de *préemption* : l'intégration de la possibilité de diviser la charge - la *préemption de charge* - et l'acheminement de la charge avec escales ou opération de transbordement - la *préemption de véhicules* -. Ces variantes, qui reflètent des hypothèses sur le fonctionnement du système, permettent d'augmenter le nombre de solutions mais aussi de considérer des chargements plus importants que la capacité du véhicule elle-même.

## 2.4 Méthodes de résolution du DARP

Nous séparons ici l'aspect statique où l'ensemble des demandes est connu en amont de la résolution et l'aspect dynamique où les demandes arrivent alors que tournées s'effectuent. L'état de l'art sur les méthodes de résolution des problèmes DARP préemptifs vient ensuite.

**Le DARP statique.** L'idée de transport à la demande est apparue dans les problématiques de Recherche Opérationnelle dès les années 70. Le problème qui en découle, le DARP, connaît un nombre important de variantes liées au contexte et n'est pas encore attaché à un unique modèle bien défini. Il est cependant important de noter que, sauf exception, toutes les variantes sont NP-difficiles.

[Psaraftis and Harilaos (1980)] proposent une des premières formulation du DARP sans y intégrer de fenêtres de temps. L'objectif est ici de minimiser le poids de chaque tournée et la non-satisfaction des usagers. Ce dernier critère se base sur les temps d'attente passés sur un nœud avant que le service ne commence. Le problème avec fenêtres de temps fut alors noté DARPTW - ang. *Dial-a-Ride problem with Time Windows* - mais l'intégration de ces contraintes s'est généralisée et le nom du problème est redevenu DARP.

La littérature recense plusieurs méthodes exactes pour résoudre le problème (bien qu'inexploitables dans un cas concret). Celles qui ont rencontré le plus de succès, sont généralement à base de programmation dynamique ([Psaraftis (1983)], [Desrosiers et al. (1986)] et [Chevrier et al. (2006)]). Ainsi [Desrosiers et al. (1986)] ont résolu des instances à 40 demandes satisfaites par un unique véhicule. Il existe d'autres méthodes comme la génération de colonnes qui est souvent utilisée dans la résolution des problèmes de transports ([Desrosiers et al. (1995)]) ou encore le *Branch-and-Cut* [Cordeau (2006)]. Mais, la complexité du problème fait que l'on a tendance à utiliser des méthodes approchées, surtout lorsqu'est envisagée une exploitation réelle où le système doit répondre rapidement aux requêtes.

Le DARP peut se décomposer en trois problèmes : un *Clustering Problem*, un *Routing Problem* et un *Scheduling Problem*, respectivement un problème de partitionnement, un problème de routage et un problème d'horodatage (calculs des dates de rendez-vous avec les usagers). La résolution de ces trois problèmes peut se faire soit indépendamment soit par un même algorithme ([Garaix (2007)]). Les résolu-

tions des deux premiers problèmes sont les plus critiques et l'ordre peut s'inverser : une première tournée géante est créée (routage) avant d'être partagée sur l'ensemble des véhicules (affectation) [Leclaire (2009)].

[Bodin and Sexton (1986)] résolvent le problème par une première phase de *clustering* qui distribue les demandes sur les différentes tournées. Cette distribution regroupe les requêtes semblables ou proches en temps et en espace. Une seconde phase réalise des échanges entre les tournées avant de résoudre le problème de routage les concernant. L'heuristique est testée sur des instances réelles de la ville de Baltimore avec 85 requêtes par instance. [Dumas et al. (1989)] ont repris cette phase de *clustering* pour résoudre à leur tour des instances de villes canadiennes. Il existe bien d'autres travaux commençant la résolution du DARP par cette distribution des demandes aux tournées comme [Desrosiers et al. (1991)], [Ioachim et al. (1995)] ou [Borndörfer et al. (1999)] qui créent un ensemble de *clusters* puis optimisent de telle sorte que les véhicules ne soient jamais vides. Dans cette étude, une fois qu'un nombre important de *clusters* est construit, un sous-ensemble est recherché afin que les demandes ne soient satisfaites qu'une seule fois (- ang. *set partitioning problem* -). Ensuite, une phase d'énumération et de sélection des tournées est réalisée.

Les métaheuristiques à base de Recherche Locale sont également employées dans la résolution du problème de Transport à la Demande. De manière itérative, un algorithme de Recherche Locale considère une solution courante, puis tente de la remplacer par une seconde de son voisinage et de meilleure performance. [Parragh et al. (2010)] mettent au point une méthode de Recherche Locale au sein d'une VNS (*Variable Neighborhood Search* de [Mladenović and Hansen (1997)]). Une heuristique établissant trois types de voisinage est alors établie. Le premier voisinage est construit par de simples opérations d'échange, le second basé sur l'idée de la chaîne d'éjection - ang. *ejection chain* - de [Glover (1996)], et le troisième exploitant les moment ou les véhicules sont à vide. D'autres travaux ont traité de la Recherche Locale que ce soit pour le DARP ou des problèmes similaires ([Healy and Moll (1995)]).

[Cordeau and Laporte (2003)] utilisent la Recherche Tabou - ang. *Tabu Search* - dans les problèmes de *Dial-a-Ride*. Comme pour la Recherche Locale, cette métaheuristique va considérer une première solution et tentera de l'améliorer par une recherche dans le voisinage de la solution. La différence principale réside dans la sauvegarde d'une suite de solutions obtenues, nommée Liste Tabou, qui permet d'éviter les cycles. Ils débutent par la recherche d'une première solution qui ne respecte que les contraintes de précédence et le fait que les dépôts commencent et terminent chaque tournée. Il peut donc ici y avoir violation des autres contraintes. Chacune des violations "autorisées" est remontée dans la fonction Objectif sous la forme d'une pénalité. Les différentes solutions sont établies dans l'exploration du voisinage se formant sur l'échange des demandes entre les véhicules. Chaque réinsertion est réalisée en minimisant l'Objectif (comprenant les critères de pénalités). La liste Tabou sauvegarde ici chaque déplacement dans le voisinage. L'heuristique réalise enfin des échanges au sein de chaque tournée dès qu'un trop grand nombre d'itérations successives n'a entraîné aucune amélioration.

Les algorithmes génétiques tiennent également une place non négligeable dans la

liste des techniques de résolution du DARP de la littérature scientifique.

[Rekiek et al. (2001)] résolvent le problème complété de contraintes relatives à l'incompatibilité entre clients. La méthode développée est basée sur l'algorithme génétique de regroupement de [Falkenauer and Delchambre (1992)]. Cette technique applique des opérateurs combinant des solutions. Ces dernières sont considérées comme performantes si elles minimisent le nombre de véhicules tout en maximisant la qualité de service. [Velasco et al. (2006)] résolvent le PDPTW (très proche du DARP) en utilisant l'algorithme *Non-Dominated Sorting Genetic* utilisé pour les problèmes multi-critères. Ces critères sont ici la minimisation de la durée des tournées ainsi que la maximisation de la satisfaction des demandes les plus urgentes. Les véhicules considérés sont des hélicoptères.

La majorité des algorithmes résolvant le DARP sont à base d'insertions successives de demandes au sein des tournées. [Jaw et al. (1986)] ont appliqué parmi les premiers une de ces méthodes au DARP. Ces travaux considèrent des demandes aller et retour satisfaites par plusieurs véhicules. Une fenêtre de temps est alors considérée à l'aller au point d'origine puis au retour au point de destination. Un temps maximum pour satisfaire chaque requête est également fourni par les usagers. Les véhicules les transportant n'ont ici pas le droit d'être à l'arrêt hormis lorsqu'un service est effectué (les temps d'attente - ang. *waiting time* - doivent être nuls). L'heuristique d'insertions successives sélectionne les demandes selon les points de ramassage les plus proches et les insère graduellement dans les différentes tournées. Leur méthode a permis de résoudre des instances générées allant jusqu'à 250 utilisateurs. Dans le même ordre d'idées, [Garaix et al. (2007)] formulent un coût à chaque insertion basé sur le prix au kilomètre et la distance parcourue. Les paramètres d'insertion sont choisis en fonction du plus petit de ces coûts, ceci est précédé par un premier tri des demandes sur les bornes MIN des fenêtres de temps. L'heuristique évalue ensuite le gain au niveau du coût que gagneraient les tournées en supprimant les demandes. Ce gain induit un second tri des demandes ciblant les opérations à effectuer lors d'une phase de recherche locale. [Diana and Dessouky (2004)] étudient la myopie des heuristiques d'insertions successives. En effet, que ce soit dans un contexte statique (demandes déterminées) ou dans un contexte dynamique, les insertions sont réalisées selon l'état courant des tournées et non des futures requêtes à insérer. [Lu and Dessouky (2006)] introduisent le concept de réalisabilité dans ces problèmes de transports. Ils mesurent l'espace temps laissé au futures requêtes et incluent cette mesure dans le choix de la demande à insérer. Le succès des techniques d'insertion naît de leur flexibilité. Elles peuvent entre autres être facilement adaptées à des cas spécifiques. [Wong and Bell (2006)] les ont mises en place pour résoudre un problème de transport à mobilité réduite nécessitant du matériel particulier, du lit au fauteuil roulant. Ils séparent différents types d'utilisation de l'espace des véhicules : les places assises et la place dévolue à tel ou tel équipement. La résolution de [Diana and Dessouky (2004)] est alors adaptée à ces spécificités pour résoudre le problème. Toujours à propos de ces méthodes, [Luo and Schonfeld (2007)] trient les demandes selon les bornes MIN des fenêtres de temps des nœuds de ramassage puis les insèrent. Ensuite, ils mettent au point un processus de réinsertion d'une demande qui,

dans l'état courant des tournées, n'a plus la possibilité d'être acceptée. Ce processus cherche une demande déjà insérée semblable à celle rejetée pour l'extraire. Si la demande rejetée peut alors s'insérer dans la tournée modifiée, le schéma classique d'insertion est alors appliqué à la demande extraite.

Plusieurs instances tirées de données réelles ont été résolues. Certaines sont pourvues de contraintes supplémentaires inhérentes aux systèmes. [Toth and Vigo (1996)] résolvent des demandes d'un système de transport de personnes handicapées. L'ensemble des demandes étant déterminé avant la résolution, c'est le DARP statique qui est résolu. Ces instances sont caractérisées par des fenêtres de temps sur chaque point et chaque destination. La spécificité réside ici dans l'homogénéité de la flotte de véhicules. Elle rassemble des minibus, des voitures spécialisées et, si les premiers ne peuvent pas accepter certaines demandes, des taxis. Le recours à ces derniers est pénalisé. La méthode de résolution est une heuristique parallèle d'insertion. Une fois qu'une solution est obtenue, celle-ci est améliorée par l'application d'opérateurs de permutations intra-tournées et inter-tournées. Il existe plusieurs travaux considérant des flottes homogènes comme [Parragh (2011)] ou encore [Borndörfer et al. (1999)] qui résolvent le problème de transport à mobilité réduite sur une flotte berlinoise (le Telebus, cf. figure 2.10) à l'aide d'une technique de *clustering*.


FIGURE 2.10 – Télébus de Berlin

La littérature scientifique du DARP révèle aussi d'autres techniques de résolution plus singulières comme [Calvo et al. (2007)] qui considèrent un graphe différent des précédents. En effet, les demandes ne sont plus représentées par un couple de nœuds mais par un seul. Chaque arc les reliant sont dotés d'un poids combinant les distances, les fenêtres de temps et les temps d'attente. Le graphe permet de déterminer le nombre minimal de véhicules nécessaire à la satisfaction de l'ensemble des requêtes.

**Le DARP dynamique.** [Psaraftis and Harilaos (1980)] considéraient déjà le problème dynamique mais sans fenêtres de temps et avec un seul véhicule considéré. Les requêtes arrivent dynamiquement sans que le système ait une quelconque information sur celles-ci. A chaque nouvelle demande, les tournées sont à nouveau recalculées sur les requêtes non encore satisfaites et en y intégrant la nouvelle re-

quête. Comme pour la très grande majorité des travaux, ils ne prennent pas en compte le fait qu'une date précise, c'est à dire le rendez-vous synchronisant véhicule et usager, peut être fixée.

Il n'est pas surprenant de constater la prédominance des techniques d'insertions successives pour intégrer des demandes dynamiques lorsque les tournées sont en train d'être exécutées. [Madsen et al. (1995)] reprennent l'heuristique de résolution à base d'insertions de [Jaw et al. (1986)] et l'adaptent au contexte dynamique. Leur solution prévoit l'intégration de chaque nouvelle demande, dans l'état courant des tournées, en moins d'une seconde. Leurs travaux s'accordaient avec un problème réel lié à un service pour personnes âgées et/ou handicapées à Copenhague. Les usagers fournissent une seule fenêtre de temps à l'origine ou à la destination. Différents types de véhicules sont utilisés. Ils ne sont pas disponibles en même temps et peuvent tomber en panne. [Colorni and Righini (2001)] proposent également une solution à base d'insertions successives, leur mécanisme fait alterner deux phases : la phase de *clustering* et la phase de routage, cette dernière étant à base de *Branch-and-Cut*. [Coslovich et al. (2006)] traitent encore de ce type de résolution dans un contexte dynamique, ils intègrent des demandes dans des tournées en cours de réalisation. Ils ont établi une mesure de la non satisfaction des usagers et ont fixé un objectif mono-critère : sa minimisation. Les instances traitées considèrent 25 à 50 demandes. Des insertions, toujours, sont au cœur des travaux de [Teodorovic and Radivojevic (2000)], l'objectif traité est proche de la somme des trois critères pondérés proposés dans ce manuscrit, à la différence près que les durées des tournées sont remplacées par la distance parcourue. Une nouvelle demande est intégrée dans une tournée suivant une série de règles. Les demandes sont évaluées selon leur incidence sur chaque critère de performance (hors temps de connexion) impliquant ensuite une seconde évaluation sur le choix du véhicule. Une dernière procédure est utilisée pour connaître l'emplacement de la demande dans la tournée sélectionnée qui implique le plus petit coût.

De même que pour le cas statique et la méthode de [Cordeau and Laporte (2003)], la recherche Tabu semble être également une technique efficace de résolution dans un contexte dynamique comme le montrent les travaux de [Mitrović-Minić et al. (2004)], [Attanasio et al. (2004)], [Berbeglia et al. (2012)] et [Kergosiena et al. (2011)].

[Mitrović-Minić et al. (2004)] forment un voisinage au moyen des chaînes d'éjection. A chaque lancement du processus d'optimisation, la résolution insère les nouvelles demandes. Ils considèrent l'impact des ces insertions à court et long terme. Ensuite, ils optimisent à nouveau l'insertion des demandes déjà intégrées aux tournées et dont le nœud *origine* n'a pas encore été traversé. Remarquons que ce processus ne prend donc pas en compte un éventuel rendez-vous avec l'usager. Ces insertions sont réalisées selon le plus bas coût. L'ordre suivi se fait selon la borne MIN la plus proche à l'origine et la plus longue distance à parcourir pour rejoindre directement la destination. Après avoir comparé différentes parallélisations de la recherche Tabu pour le problème DARP statique, [Attanasio et al. (2004)] intègrent l'implémentation de [Cordeau and Laporte (2003)] dans le problème dynamique accompagnée d'une procédure *infinite penality* validant les différentes solutions obtenues sur chaque unité

de calcul. [Berbeglia et al. (2012)] reprennent également la base de [Cordeau and Laporte (2003)] en l'intégrant au cas dynamique, mais, cette fois-ci, la recherche Tabu s'accompagne d'une procédure exacte de programmation par contrainte. [Kergosiena et al. (2011)] exploitent la recherche Tabu dans un contexte réel : le transport de patients.

La résolution du DARP dynamique et l'évaluation de ses solutions sont également réalisées par simulation. [Fu (2002)] en a décrit un modèle permettant l'évaluation de services de transport à la demande. Le degré d'abstraction est peu élevé (par exemple l'outil va jusqu'à intégrer les technologies de communication entre les différentes entités du système) ce qui amène ces travaux à conclure sur la difficulté du transfert d'expérience d'un site à un autre (ceci incluant la partie management de flotte de véhicules). [Grötschel et al. (1999)] se sont basés sur la librairie AMSEL, une librairie en langage C de simulation par évènements, pour développer leur modèle. Leur travail consiste à tester de simples techniques d'insertion de requêtes dynamiques comme l'insertion selon l'ordre d'arrivée (FIFO), l'insertion avec nouvelle planification des demandes non servies (les rendez-vous avec l'usager sont donc recalculés) et l'insertion en fin de planning. [Noda et al. (2003)] mettent au point une simulation permettant d'évaluer deux types de transports : les services statiques (horaires fixes) et les services de transports à la demande (horaires variables). Cet outil permet de comparer les deux types de service selon le nombre de demandes satisfaites par le système selon le temps entre la réalisation de la requête et l'émission de celle-ci.

Enfin, [Borndörfer et al. (1999)] indiquent que, dans littérature scientifique relevant du DARP, la distinction entre sa version statique et sa version dynamique reste floue. Plusieurs élément qui devraient intervenir dans le modèle dynamique "pur" manquent, par exemple l'intégration d'annulation des requêtes par les usagers. Comme nous l'avions remarqué sur quelques exemples, les services proposent la plupart du temps de réorganiser les tournées à l'arrivée de nouvelles demandes sans tenir compte de rendez-vous pris avec les usagers. Ces rendez-vous sont pourtant primordiaux pour synchroniser véhicules et usagers. La principale difficulté réside dans le fait que les tournées sont peu flexibles une fois qu'une première série de demandes est intégrée. Les futures requêtes sont alors plus difficiles à insérer au détriment de l'exploitant et de l'usager. Ils concluent, comme [Noda et al. (2003)], sur la difficulté de transfert d'expérience d'un site à l'autre.

Remarquons tout de même que certaines techniques de résolution du DARP dynamique sont déjà utilisées dans de réelles exploitations de transport à la demande, en milieu urbain, de personnes âgées ou handicapées. Par ailleurs, nous pouvons dénombrer bon nombre d'utilisations dans le cadre hospitalier (transports de patients mais aussi de matériel, greffes et médicaments, très contraints en temps). Il est possible de se référer à [Beaudry et al. (2012)] qui relèvent les exemples les plus pertinents. De manière plus générale, [Cordeau and Laporte (2007)] font l'objet d'un état de l'art poussé sur les solutions au *Dial-a-Ride* statique mais aussi dynamique et plus récemment [Pillac et al. (2013)] se concentrent sur la littérature des problèmes dynamiques plus généraux : les DVRP - ang. *Dynamic Vehicle Routing Problem* -.

**La préemption de charge.** Le fractionnement de la demande ou préemption de chargement permet d'améliorer les performances dans les différents systèmes de transports. Il est même nécessaire dès qu'une demande nécessite une capacité supérieure à celle des véhicules. Par rapport aux problème classique, la préemption de chargement se doit d'améliorer les solutions dans les problèmes d'optimisation associés tels que celui du ramassage (PRL) et de sa variante "avec fractionnement" (PRLF) où la somme des parties d'une demande doit être égale au chargement initial. Les solutions obtenues montrent que le PRLF est plus efficace que le PRF mais ceci s'explique par le fait que le premier est en réalité une relaxation du second lorsque le problème est sous la forme d'un problème linéaire. Pour la plupart des problèmes qui sont difficiles, cette relaxation n'implique pas forcément une recherche de solution plus aisée. D'autres problèmes de recherche opérationnelle et d'optimisation combinatoire incluant le fractionnement ont été plus largement étudiés dans les problèmes de tournées de véhicules. Mais, aucune étude scientifique n'introduit la division du chargement des demandes dans un problème de transport à la demande. La littérature est maigre dans le domaine (et donc nulle en ce qui concerne le DARP avec préemption de charge). Cet état de l'art se concentre donc sur les problèmes les plus proches.

[Nowak et al. (2008)] ont travaillé sur un PRLF mono-véhicule avec une capacité non nécessairement supérieure au chargement des demandes. Leur solution se décompose en deux étapes. La première consiste en une première solution du PRL et la deuxième est une solution du PRLF en itérant sur un processus de fractionnement des demandes et en se basant sur la solution initiale. Ils ont montré que le gain obtenu par le fractionnement était maximisé lorsque le volume de chaque demande était supérieur à la moitié de la capacité des véhicules qui les accueillent. Ceci paraît logique puisque, sans division possible, les solutions ne traduisent que l'acheminement d'un seul chargement à la fois. Ils ont relevé un gain de performance de 40% par cette méthode avec un unique véhicule mais, dès lors qu'ils ont considéré le cas multi-véhicules, ce gain devient négligeable en terme de coût. En effet, malgré une baisse du nombre d'engins motorisés qui seraient nécessaires à la satisfaction de l'ensemble des requêtes, la performance recherchée dans leurs travaux n'implique pas de diminution du nombre de véhicules.

[Dror and Trudeau (1989)] sont à l'origine des premiers travaux sur le cas des tournées de véhicules avec fractionnement (PTVF) en développant une métaheuristique. Les instances traitées considèrent chaque chargement de chaque demande de volume inférieur ou égal à la capacité et la population de véhicules disponibles ne connaît pas de limite. Leur processus de résolution débute sans considérer de fractionnement puis améliore la solution trouvée par diverses techniques de voisinage les intégrant. Ils ont fait ressortir que, si les demandes ont des chargements suffisamment grands, le fractionnement de la demande améliore le coût jusqu'à 14% et diminue le nombre de véhicules jusqu'à 25%, le tout obtenu sur des volumes de demandes de 75 à 150.

Nous avons vu que la métaheuristique Recherche Tabu ([Glover (1990)]) était efficace pour la résolution du DARP ([Cordeau and Laporte (2003)]). Elle est éga-

lement utilisée pour le PTVF par [Archetti et al. (2006)]. Ils traitent d'instances où les requêtes des usagers portent sur l'acheminement d'un volume parfois supérieur à ce que peut transporter un véhicule. Le traitement part d'une première solution et distribue des parties du chargement des demandes, celles déjà insérées dans les tournées, jusqu'à ce que le volume transporté par les véhicules devienne inférieur à la capacité. Une bibliographie des PTVF plus complète a été établie par [Archetti and Speranza (2008)].

**La préemption de vehicule.** Les problèmes de *Dial-a-Ride* avec fenêtres de temps ont rarement fait l'objet de recherches lorsqu'il s'agit d'y intégrer la possibilité de transborder le chargement d'une demande d'un véhicule à un autre lors de la prise en charge de celle-ci. L'intégration des transbordements - ang. *transhipment* - (ou encore parfois noté transferts - ang. *tranfers* - quand ce n'est pas préemption de véhicule - ang. *vehicle pre-emption* -), a plus longuement fait l'objet de travaux de recherche dans les problèmes de tournées de véhicules classiques aux contraintes moins importantes (comme par exemple la non prise en compte des fenêtres de temps ou des durées maximum de connexion). Les auteurs de [Masson et al. (2011)] et [Masson et al. (2012)] ont vraisemblablement été les seuls à traiter le problème de *Dial-a-Ride* avec transferts qu'ils notent DARPT. Comme [Cordeau and Laporte (2003)] pour le DARP, ils proposent une résolution à base d'une recherche Tabu en minimisant les distances, ils proposent également une procédure évaluant la faisabilité d'un problème. Il faut noter qu'ici, les points relais, ou points de transfert, sont déterminés en amont de la résolution, ils ne sont pas créés dynamiquement le long de l'exécution. Ensuite, seules les études relatives au *Pickup and Delivery Problem with Transfers* (PDPT) semblent s'approcher réellement de la problématique étudiée dans cette partie du manuscrit.

Tout d'abord relevons les méthodes exactes résolvant le PDPT. [Kerivin et al. (2008)] posent l'ensemble des nœuds comme points de transfert compatibles. Ils présentent une formulation mathématique du problème sans fenêtre de temps mais, en plus de considérer les transferts, intègrent une relaxation sur le chargement : ce dernier peut être acheminé en plusieurs fois et par plusieurs véhicules. La résolution des instances dont le nombre maximum de requêtes se limite à 15 est résolue par la méthode *Branch-and-Cut*. [Nakao and Nagamochi (2012)] ont, quant à eux, réalisé une analyse du pire cas du PDPT en établissant une borne $Min$ du problème sans fenêtre de temps et avec un unique point de transfert. Enfin, pour clore ce paragraphe sur les méthodes exactes, relevons les travaux de [Cortes et al. (2010)] qui présentent une formulation mathématique du PDPT et le résout avec un *Branch-and-Cut*.

Les méthodes approchées sont plus à même de résoudre le problème dans les temps impartis aux systèmes des transports à la demande. [Cortés and Jayakrishnan (2002)] se proposent, au travers d'une simulation, de résoudre des instances du problème avec une large population de véhicules. Leurs heuristiques traduisent un ensemble de règles permettant la sélection du véhicule puis le calcul du traçage de sa route afin d'aiguiller la solution vers l'espace réalisable sous les contraintes d'exécution dû au temps réel. Comme [Parragh et al. (2010)] pour la résolution du

DARP sans transfert, [Masson et al. (2013)] proposent une heuristique de recherche de voisinage testée sur de larges instances allant jusqu'à 193 requêtes. [Mitrovic-Minic and Laporte (2006)] ont également testé leur algorithme sur des instances de grande taille (100 requêtes) en mettant au point une Recherche Locale pour la résolution du PDPT sans capacité. [Shang and Cuff (1996)], puis [Thangiah et al. (2007)] dans sa version dynamique, ont réalisé une heuristique posant chaque nœud du graphe comme nœud de transfert possible, il permet de servir de transbordement dès lors que leur technique d'insertions ne trouve pas de solution pour une intégration classique d'une demande au sein des tournées. Ces deux derniers travaux montrent encore l'efficacité des techniques heuristiques d'insertions pour la résolution de problèmes de tournées difficiles.

## 2.5 Les problèmes couplant transport et production

Cette thèse traite en grande partie des transports à la demande mais se conclut sur un chapitre traitant d'une problématique couplant des questions de production et de transport. En effet, ce problème particulier de *Lot-Sizing* intègre des transferts et stockages de produits une fois ces derniers fabriqués ([Sambasivan and Yahya (2005)]). A première vue, il semble y avoir peu d'intérêt à traiter dans une même étude ces deux sujets ; cependant, dans la pratique, les similitudes sont réelles. Les transferts intégrés aux problèmes de *Lot-Sizing* posent la question de l'articulation entre routage et production, les unités de production agissant alors comme les acteurs d'un DARP ou d'un PDP. D'autres exemples existent comme les problèmes de *Job Shop Scheduling* couplés à une problématique de transport ([Zhang et al. (2013)]). Ces analogies se trouvent renforcées si on considère les problèmes comme posés dans un contexte dynamique.

# Chapitre 3

# Modélisation et résolution du DARP statique

## Introduction et notations

Nous nous intéressons ici au problème *Dial-a-Ride* (DARP) standard et statique. Associé aux transports à la demande, il concerne le routage et la construction d'horaires des véhicules devant acheminer des charges depuis des nœuds *origine* vers des nœuds *destination* à travers un réseau routier. Après avoir donné le cadre formel du problème, nous présentons notre modélisation qui se base sur un réseau réduit. En effet, ce dernier considère uniquement les demandes et les dépôts de véhicules. Nous détaillons ensuite notre schéma de résolution c'est-à-dire la manière de construire les tournées. Le problème étant bien trop complexe pour être résolu de manière optimale, nous avons opté pour les méthodes heuristiques d'insertions successives afin de l'adapter plus facilement ensuite au DARP dynamique. Nous enrichissons ici les méthodes d'insertion de mécanismes de propagation de contraintes. Plusieurs batteries de tests sont réalisées afin d'étudier le fonctionnement de cette méthode (temps d'exécution, qualité de la solution etc.).

Définissons les différentes notations utilisées le long des chapitres relatifs à notre modèle du DARP. Soit $\Gamma$ une collection de $K$ listes $\Gamma_k, k = 1..K$. Nous pouvons alors définir :

- Tete($\Gamma_k$) = le premier élément de $\Gamma_k$,
- Queue($\Gamma_k$) = le dernier élément de $\Gamma_k$,
- ListeSansTete($\Gamma_k$) = la sous-liste de $\Gamma_k$ obtenue après lui avoir ôté Tete($\Gamma_k$).

Etant donné un élément $z$ de $\Gamma_k$, nous notons $Succ(\Gamma_k, z)$ l'élément successeur de $z$ dans $\Gamma_k$ et $Pred(\Gamma_k, z)$ l'élément prédécesseur de $z$ dans $\Gamma_k$. Etant donné un nœud $z'$ sur la même liste (tournée) que $z$, nous posons :

- $z \ll_{\Gamma_k} z'$ pour exprimer : $z$ est localisé avant $z'$,
- $z \ll^{=}_{\Gamma_k} z'$ pour exprimer : $z \ll_{\Gamma_k} z'$ ou $z = z'$,
- Segment($\Gamma_k$, $z$, $z'$) = la sous-liste définie par tous les éléments $z''$ tels que $z \ll^{=}_{\Gamma_k} z'' \ll^{=}_{\Gamma_k} z'$.

Nous utiliserons d'ailleurs l'opérateur '=' exclusivement en tant que comparateur, l'affectation étant notée ←, lors de l'écriture des algorithmes.

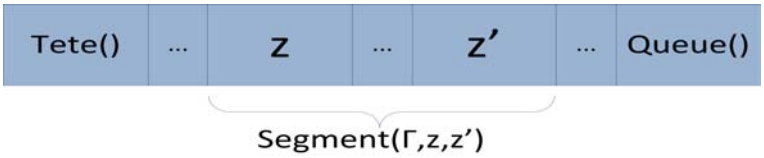


FIGURE 3.1 – Exemple de Segment dans une liste

## 3.1 Le modèle DARP statique

### 3.1.1 Réseau réel et réseau réduit

Un problème DARP, posé dans son environnement réel, c'est-à-dire par rapport à un réseau de transport réel parcouru par ses véhicules et usagers, est principalement défini par :

- un graphe $G = (V, E)$ modélisant un **réseau de transport**. Ce graphe contient un nœud nommé *Dépôt*. $V$ contient également deux sous-ensembles $DE$ et $AR$ désignant respectivement les nœuds *origine* et les nœuds *destination* des demandes. Les arcs $e \in E$ sont dotés de temps de parcours $l(e)$ positifs ou nuls.
- une flotte de $K$ véhicules nommée $VH$ d'index $k$, $k = 1..K$. Ces derniers sont limités par un chargement maximum $CAP$ ;
- un ensemble $D$ de demandes $D_i, i = 1..|D|$ où chaque $D_i$ (ou par extension de l'index, "chaque demande $i$") a les 6 composantes ($o_i$, $d_i$, $\Delta_i$, $F(o_i)$, $F(d_i)$, $Q_i$) définies de la manière suivante :
  - $o_i$ représente l'origine de $D_i$,
  - $d_i$ est la destination de $D_i$,
  - $F(o_i)$ est une fenêtre de temps qui fournit l'intervalle de temps pendant lequel l'usager veut partir,
  - $F(d_i)$ est une fenêtre de temps qui fournit l'intervalle de temps pendant lequel l'usager veut arriver,
  - $\Delta_i \geq 0$ est la durée maximum pour satisfaire $D_i$,
  - $Q_i$ exprime le chargement relatif à $D_i$ et les opérations de chargement puis de déchargement se traduisent sur les nœuds *origine* puis *destination* par $Q_{o_i} = Q_i = -Q_{d_i}$ ;
- une borne $\Delta^k$ sur la durée maximum de la tournée du véhicule $k$.

Le problème consiste alors à construire des tournées, pour les véhicules de la flotte $VH$, qui répondent aux différentes demandes en satisfaisant contraintes temporelles et contraintes de capacité, et qui minimisent une forme de coût. Nous simplifions les données ci-dessus en nous affranchissant du réseau réel et en le remplaçant par un

**réseau réduit**. Pour ce faire, nous restreignons le modèle aux seuls nœuds présents dans D (un nœud pour chaque $o_i$ et un autre pour chaque $d_i$), et en utilisant le fait que tous les véhicules calculent leur propre route selon la stratégie du **plus court chemin**. Remarquons que des points géographiquement identiques peuvent se retrouver dissociés dans ce réseau réduit et séparés par une distance nulle. Nous ajoutons, pour chaque véhicule, deux nœuds de Dépôt $DepotD_k$ (départ du Dépôt) et $DepotA_k$ (arrivée au Dépôt).

Les entrées du DARP standard, ainsi reformulées, sont :
- l'ensemble des $K$ véhicules $k$ de la flotte $VH$,
- leur capacité commune $CAP$,
- l'ensemble des nœuds $X = \{DepotD_k,\ DepotA_k,\ k = 1..K\} \cup \{o_i,\ d_i$ avec $i = 1..|D|)$,
- la matrice des distances (sous forme de temps) DIST où pour tout couple de nœuds $(x\,;\ y)$, DIST$(x,y)$ est égale à la longueur du plus court chemin connectant $x$ et $y$, la matrice satisfaisant l'inégalité du triangle.

Chaque nœud $x$ de $X$ est caractérisé par :
- son statut tel que $Statut(x) \in \{Origine,\ Destination,\ DepotD,\ DepotA\}$, nous posons $Depot = DepotD \cup DepotA$,
- l'index Dem de la demande associée dans le cas où $x$ est l'origine ou la destination d'une demande : $\text{Dem}(x) = i$ : si $x = o_i$ ou $d_i$ et $\text{Dem}(x) = 0$ sinon,
- l'index $VI$ du véhicule $k$ si $x$ est le départ ou l'arrivée d'une tournée ($VI(x) = 0$ pour tous les autres nœuds),
- la quantité de *chargement* $Q_x$ définie par :
  - si $Statut(x) \in Depot$ alors $Q_x = 0$,
  - si $x$ est l'origine d'une demande $i$, alors $Q_x = Q_i$,
  - si $x$ est la destination d'une demande $i$, alors $Q_x = -Q_i$ ;
- son nœud homologue $Jumeau(x)$ :
  - si $x = DepotA_k$ alors $Jumeau(x) = DepotD_k$ et réciproquement,
  - si $x = o_i$ alors $Jumeau(x) = d_i$ et réciproquement ;
- sa fenêtre de temps $F(x)$ : pour toute tournée $k$ dans $K$, $F(DepotA_k) = [0, +\infty[ = F(DepotD_k)$ ; si $x = o_i$ ou $d_i$, $F(x)$ est la fenêtre de temps imposée à $x$ dans la description de l'ensemble $D$ ; on pose $F(x) = [F.\min(x),\ F.\max(x)]$ ;
- sa borne de transit $\Delta(x)$ définie par : si $x = o_i$ ou $d_i$, alors $\Delta(x) = \Delta_i$, sinon $\Delta(x) = \Delta^k$ avec $\Delta^k$ borne supérieure correspondante à la durée maximale de la tournée $k$.

**Homogénéité des véhicules.** Le cas général du *Dial-a-Ride Problem* postule l'homogénéité des véhicules, que ce soit sur les coordonnées des dépôts ou encore la capacité des véhicules. Toutefois, on pourra se rendre compte que tant les modèles proposés ici que les schémas algorithmiques mis en œuvre permettent assez facilement de s'affranchir de cette hypothèse. De la même façon, notre *framework* nous permettrait également de différencier les durées maximales pour chaque tournée.

**Expression d'une solution au DARP.** Nous pouvons maintenant résumer le fonctionnement du système une fois qu'une solution réalisable est trouvée. Chaque véhicule $k$ de $K$ débute sa tournée en $DepotD$ au moment $t(DepotD)$ et le termine en $DepotA$ à $t(DepotA)$. Pour chaque demande $i$ couverte par $k$, le véhicule $k$ arrive en $o_i$ au moment $t(o_i) \in F(o_i)$, charge $Q_i$ pour le transporter jusqu'à $d_i$ au temps $t(d_i) \in F(d_i)$ où il décharge $(-Q_i)$. On doit alors avoir : $t(d_i) - t(o_i) \leq \Delta_i$. La résolution d'une instance ($X$, DIST, $K$, $CAP$) du DARP standard signifie donc le calcul de ces tournées $\Gamma_k, k = 1..K$, et des temps de passage $t(x)$, $x \in X$, associés. Ce calcul doit être fait de telle sorte que la performance économique ainsi que la qualité de service soit la plus élevée possible. Afin de qualifier un tel ensemble de tournées comme une solution réalisable au problème DARP standard et statique, nous devons préciser la nature de ces objets "tournées" et des contraintes qui leur sont imposées.

### 3.1.2 Validité d'une Tournée et Collection de Tournées datées

Soit un parcours $\Gamma_k$ dans $X$, correspondant au véhicule $k$ : $\Gamma_k$ est donc une liste, débutant avec le nœud $DepotD_k$ et s'achevant avec le nœud $DepotA_k$. $\Gamma_k$ sera nommée une *tournée*, si, en plus de cette condition sur son début et sa fin, on a aussi :

— pour tout nœud $x = o_i$ (respectivement $d_i$) de $\Gamma_k$, le nœud $Jumeau(x)$ est nécessairement dans $\Gamma_k$, et nous avons $x \ll_{\Gamma_k} Jumeaux(x)$ (respectivement $Jumeaux(x) \ll_{\Gamma_k} x$) (*Contraintes de précédence et de couplage*),

— pour tout nœud $x$ dans $\Gamma_k$, si $x \neq \text{Tete}(\Gamma_k)$, et $x \neq \text{Queue}(\Gamma_k)$, alors $Statut(x) \notin Depot$.

On dira que $\Gamma_k$ est **charge valide** si et seulement si pour tout $x$ de $\Gamma_k$, $x \neq Tete()$, nous avons $\sum_{y=Tete(\Gamma_k)}^{y=x} Q_y \leq CAP$. De la même manière, la tournée $\Gamma_k$ est dite **temps valide** si et seulement si il existe un ensemble de valeurs $t(x), x \in \Gamma_k$, tel que les trois contraintes suivantes sont respectées, pour tout $x$ de $\Gamma_k$ : (E1)

— $t(Succ(\Gamma_k, x)) \geq t(x) + DIST(x, Succ(\Gamma_k, x)), x \neq Queue(\Gamma_k)$ (*Contrainte des distances*),

— $|t(Jumeau(x))–t(x)| \leq \Delta(x))$ (*Contraintes sur les durées de connexion et de tournée*),

— $t(x) \in F(x)$ (*Contraintes des fenêtres de temps*).

Un tel système de valeur $t(x)$, $x \in \Gamma_k$, sera nommé système de dates pour $\Gamma_k$. Si la tournée $\Gamma_k$ est donnée en même temps qu'un système de dates, on dit qu'il s'agit d'une tournée datée. Une tournée datée $\Gamma_k$ est dite **valide** (ou admissible) si elle est à la fois *temps valide* et *charge valide*. Une collection $\Gamma = (\Gamma_k, k \in K)$ de tournées datées sera dès lors admise, du point de vue de notre problème DARP, si chaque tournée $\Gamma_k$ est valide, et si chaque demande $D_i, i = 1..|D|$, est traitée dans exactement une tournée.

### 3.1.3 Evaluation d'une Tournée et d'une collection de Tournées

Nous allons maintenant nous intéresser à l'évaluation de la performance d'une tournée $\Gamma_k$ datée, et supposée valide. Rappelons qu'une tournée peut être jugée selon deux principaux points de vue : celui de l'opérateur et celui de l'usager. Le premier aura tendance à faire en sorte que sa flotte consomme le moins possible (ceci peut impliquer un nombre réduit de véhicules en fonctionnement) et le second espère des temps d'attente et de parcours les plus courts possibles. Le modèle DARP que nous étudions exprime ces objectifs selon trois critères de temps : la durée globale d'une tournée $\Gamma_k, k \in K$, datée selon un système de date $t = (t(x), x \in \Gamma_k)$, la somme des temps nécessaires à la satisfaction de chaque demande transportée par le véhicule $k$, et les temps d'attente sur cette tournée, ces quantités sont notées respectivement $T_{Global}(k,t)$, $T_{Ride}(k,t)$ et $T_{Wait}(k,t)$ leurs valeurs sont données par les trois formules suivantes 3.1, 3.2 et 3.3.

$$T_{Global}(k,t) = t(Queue(\Gamma_k)) - t(Tete(\Gamma_k)) \tag{3.1}$$

$$T_{Ride}(k,t) = \sum_{i \in D_{\Gamma_k}} (t(d_i) - t(o_i)) \tag{3.2}$$

$$T_{Wait}(k,t) = T_{Global}(k,t) - \sum_{x \in ListeSansTete(\Gamma_k)} DIST(Pred(x), x) \tag{3.3}$$

Ces indices de performance reflètent une réalité multicritère. Nous définissons donc la performance ou valeur d'une tournée $\Gamma_k$ à l'aide de 3 coefficients A, B, C, selon la formule 3.4.

$$Eval(k,t) = A.T_{Global}(k,t) + B.T_{Ride}(k,t) + C.T_{Wait}(k,t) \tag{3.4}$$

La somme des évaluations de toutes les tournées des $K$ véhicules nous permet de mesurer la performance du planning général de la flotte $VH$. La valeur d'une collection de tournées $\Gamma = (\Gamma_k, k = 1..K)$, datée selon un système $t = t(x), x \in X$, vient alors de façon naturelle selon la formule 3.5.

$$\begin{aligned} Eval(\Gamma,t) &= \sum_{k \in K} Eval(k,t) \\ &= \sum_{k \in K} (A.T_{Global}(k,t) + B.T_{Ride}(k,t) + C.T_{Wait}(k,t)) \end{aligned} \tag{3.5}$$

### 3.1.4 Modèle DARP Standard

Le problème DARP Standard ainsi formalisé consiste ici en la recherche d'une collection de tournées $\Gamma = (\Gamma_k, k = 1..K)$, datée selon un système de dates $t = (t(x), x \in X)$, *admissible*, et minimisant le coût Eval($\Gamma$, $t$). Les INPUT et OUTPUT du DARP Standard sont donc les suivants :

— **INPUT** :
  — le nombre $K$ de véhicules et leur capacité $CAP$,
  — l'ensemble $D$ des demandes $D_i, i = 1..|D|$,
  — l'ensemble des coefficients multicritères A, B, C, positifs,
  — l'ensemble $X$ des nœuds du réseau réduit accompagnés de leurs caractéristiques,
  — la matrice $DIST$ des distances sur $X$ ;
— **OUTPUT** :
  — une collection de tournées $\Gamma = (\Gamma_k, k = 1..K)$, et un système de dates $t = (t(x), x \in X)$ associé. Cette collection $\Gamma$ ainsi datée doit être admissible et doit minimiser la quantité $Eval(\Gamma, t) = A.T_{Global}(k, t) + B.T_{Ride}(k, t) + C.T_{Wait}(k, t)$.

**Exemple : deux objets solutions pour deux pondérations.** Définissons tout d'abord les paramètres du problème. Le nombre de demandes, la population de la flotte ainsi que la capacité de chaque véhicule sont donnés en tableau 3.1.

| | |
|---|---|
| \|D\| | 5 |
| K | 2 |
| CAP | 2 |

TABLE 3.1 – Paramètres d'un exemple sur le DARP Standard

Ensuite, les positions de chaque origine et de chaque destination sont représentées en figure 3.2, nous pouvons remarquer que les dépôts ne sont pas nécessairement disposés sur un même lieu géographique. Pour simplifier l'exemple, nous n'allons pas considérer de contrainte de temps.

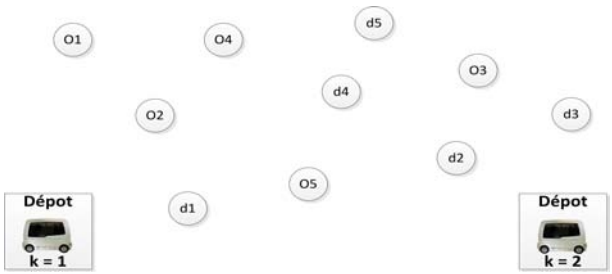


FIGURE 3.2 – Exemple de graphe pour le DARP Standard

Deux résolutions sont calculées suivant des fonctions Objectif pondérées de manière différente. La première minimise le temps global d'une tournée et la seconde le temps de connexion, le tableau 3.2 indique ces deux pondérations.

| | A | B | C |
|---|---|---|---|
| Exemple 1 | 1 | 0 | 0 |
| Exemple 2 | 0 | 1 | 0 |

TABLE 3.2 – Pondération des deux exemples sur le DARP Standard

Les figures 3.3 et 3.4 sont respectivement les solutions trouvées pour l'exemple 1 et l'exemple 2. Nous voyons d'emblée la différence : le premier cherche à rentrer les véhicules au dépôt au plus tôt et le second amène tout usager venant de monter dans le véhicule à sa destination sans étape supplémentaire.

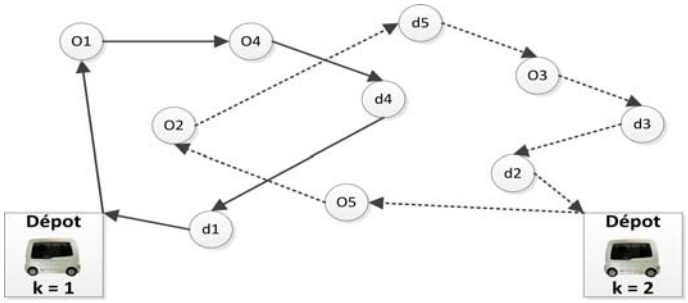


FIGURE 3.3 – Solution de l'exemple 1

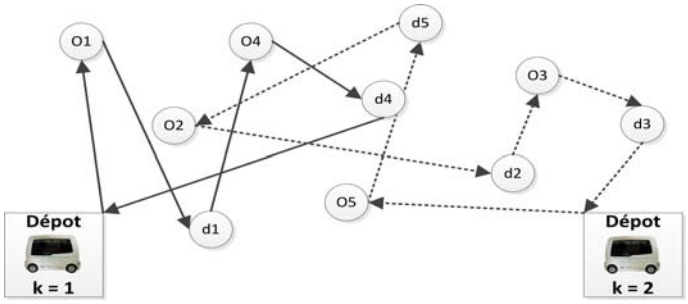


FIGURE 3.4 – Solution de l'exemple 2

**Remarque sur les temps de service.** Nous ne prenons pas ici en compte de "temps de service". Un temps de service $\delta(x)$, est supposé mesurer la durée des processus de chargement et de déchargement sur le nœud $x$. En effet, cette notion de temps de service n'a de sens que si les quantités $\delta(x)$ sont fonction de la charge et de l'état du véhicule au moment où il arrive en $x$. Dans le cas contraire, c'est à dire si les $\delta(x), x \in X$, sont fixes, ce qui est l'hypothèse adoptée dans la plupart des études, (cf. [Cordeau and Laporte (2003)] ou [Cordeau (2006)]), il suffit de reporter les temps de service dans la matrice des distances $DIST$ pour simplifier du même coup le modèle.

### 3.1.5 Le modèle mathématique adapté de [Cordeau (2006)]

Cette section reprend le modèle mathématique des problèmes de *Dial-a-Ride* avec fenêtres de temps basé sur [Cordeau (2006)] en l'adaptant à nos hypothèses. Nous notons $V' = DE \cup AR$. Les variables sont (page suivante) :

- $Z^k_{x,y}$, $k = 1..K$, $x, y \in X$, à valeurs dans $(0, 1)$ et de sémantique : $Z^k_{x,y} = 1$ si $x$ et $y$ sont consécutifs dans $\Gamma_k$, 0 sinon ;
- $ChT^k_x$, $k = 1..K$, $x \in X$, à valeurs dans $\mathbb{Q}$, positives, et de sémantique : $ChT^k_x$ est la charge du véhicule $k$ quand celui-ci sort du nœud $x \in \Gamma_k$ ($ChT^k_x = 0$ si $x \notin \Gamma_k$) ;
- $t^k_x$, $k = 1..K$, $x \in X$, à valeurs dans $\mathbb{Q}$, positives, et de sémantique : $t^k_x$ est la date de passage du véhicule $k$ au nœud $x$, quand $x \in \Gamma_k$ ($t^k_x = 0$ si $x \notin \Gamma_k$).

Fonction Objectif :

$$\begin{aligned} Min[&A.\sum_k (t^k_{DepotA_k} - t^k_{DepotD_k}) + B.\sum_i (t^k_{d_i} - t^k_{o_i}) \\ &+C.\sum_k (t^k_{DepotA_k} - t^k_{DepotD_k} - \sum_{x,y} DIST(x,y) Z^k_{x,y})] \end{aligned} \tag{3.6}$$

Sous les Contraintes :

$$\sum_{k \in K} \sum_{y \in V'} Z^k_{x,y} = 1, \forall x \in V' \tag{3.7}$$

$$\sum_{y \in V'} Z^k_{x,y} - \sum_{y \in V'} Z^k_{|D|+x,y} = 0, \forall x \in DE, k \in K \tag{3.8}$$

$$\sum_{y \in DE} Z^k_{0y} = 1, \forall k \in K \tag{3.9}$$

$$\sum_{y \in V'} Z^k_{y,x} - \sum_{y \in V'} Z^k_{x,y} = 0, \forall x \in V', k \in K \tag{3.10}$$

$$\sum_{x \in V} Z^k_{x,2|D|+1} = 1, \forall k \in K \tag{3.11}$$

$$t^k_y \geq (t^k_x + DIST(x,y)) Z^k_{x,y}, \forall x \in V, y \in V, k \in K \tag{3.12}$$

$$ChT^k_y \geq (ChT^k_x + Q_y) Z^k_{x,y}, \forall x \in V, y \in V, k \in K \tag{3.13}$$

$$DIST(x, |D| + x) \leq t^k_{|(D)|+x} - t^k_x \leq \Delta(x), x \in DE \tag{3.14}$$

$$F.min(x) \leq t_x^k \leq F.max(x), \forall x \in V, k \in K \tag{3.15}$$

$$t_{2|D|+1}^k - t_0^k \leq \Delta^k, \forall k \in K \tag{3.16}$$

$$ChT_x^k \leq CAP^k \tag{3.17}$$

$$Z_{x,y}^k \in \{0;1\}, t \in R^+ \tag{3.18}$$

**Remarques.** La quantité à minimiser choisie dans [Cordeau (2006)] porte sur la somme des longueurs des tournées : $DIST(x,y)Z_{x,y}^k$. Aussi, le programme linéaire utilise ici une formulation à trois index mais il existe bien d'autres formulations dans la littérature dont même d'autres à deux index ([Cordeau and Laporte (2007)]).

## 3.2 Procédés de résolution

### 3.2.1 Gestion des contraintes et évaluation des insertions

Une fois le problème DARP posé, nous allons étudier sa résolution au travers de diverses techniques d'insertion. Nous partirons d'une tournée $\Gamma_k$ dans laquelle le processus tentera d'insérer des demandes $D_i$ tout en maintenant sa validité (au niveau des contraintes). Une tournée $\Gamma_k$ courante étant donnée, l'insertion d'une nouvelle demande $D_i$ s'effectue selon 2 paramètres d'ancrage $x$ et $y$ dans $\Gamma_k$ :

— $o_i$ est inséré entre $x$ et $Succ(\Gamma_k, x)$,
— $d_i$ est inséré entre $y$ et $Succ(\Gamma_k, y)$,
— $d_i$ est inséré après $o_i$ dans $\Gamma_k$.

#### Contraintes de charge

Une tournée $\Gamma_k$ est *charge valide* si pour tout $x$ dans $\Gamma_k$, l'inégalité $\sum_{y=Tete(\Gamma_k)}^{y=x} Q_y \leq CAP$ est respectée. Les véhicules partent à vide du dépôt. On maintient à jour, en même temps que les tournées $\Gamma_k, k = 1..K$, les quantités $ChT(\Gamma_k, x), x \in \Gamma_k$, telles que : $ChT(\Gamma_k, x) = \sum_{y=Tete(\Gamma_k)}^{y=x} Q_y$. Il est simple de vérifier si l'insertion de la demande $D_i, i = 1..|D|$, dans $\Gamma_k$ implique ou pas un dépassement de la capacité, c'est-à-dire en testant l'impact de l'ajout de la charge $Q_i$ sur les quantités $ChT(\Gamma_k, z)$, pour $z$ allant dans $\Gamma_k$ de $Succ(\Gamma_k, x)$ à $y$, $x$ et $y$ étant les deux paramètres d'ancrage de l'insertion de $D_i$.

#### Contraintes de temps

Nous gérons les contraintes de temps par propagation de contraintes, en maintenant à jour, pour chaque nœud $x$ d'une tournée $\Gamma_k$, sa fenêtre de temps courante $F(x) = [F.\min(x)\,;F.\max(x)]$. Ces fenêtres de temps $F(x)$, $x \in \Gamma_k$, sont contractées au fur et à mesure des insertions. Plus précisément, nous propageons, à partir de toute tentative d'insertion dans $\Gamma_k$ d'une demande $D_i$, les règles d'inférence suivantes (illustrations 3.6 et 3.5) :

— règles relatives aux distances :
  — **R1** :
  $$FP.min(x) + DIST(x,y) > FP.min(y)\,;\; y = Succ(\Gamma_k, x)\,;$$
  $$\models$$
  $$(FP.min(y) \leftarrow FP.min(x) + DIST(x,y))$$
  — **R2** :
  $$FP.max(y) - DIST(x,y) < FP.max(x)\,;\; y = Succ(\Gamma_k, x)\,;$$
  $$\models$$
  $$(FP.max(x) \leftarrow FP.max(y) - DIST(x,y))$$

— règles relatives aux bornes $\Delta_i, i = 1..|D|$, et $\Delta^k$ :
  — **R3** :
  $$FP.min(x) < FP.min(y) - \Delta(x)\,;\; y = \text{Jumeau}(\Gamma_k, \text{x})\,;\; x \ll_{\Gamma_k} y\,;$$
  $$\models$$
  $$(FP.min(x) \leftarrow FP.min(y) - \Delta(x))$$
  — **R4** :
  $$FP.max(y) > FP.max(x) + \Delta(x)\,;\; y = \text{Jumeau}(\Gamma_k, \text{x})\,;\; x \ll_{\Gamma_k} y\,;$$
  $$\models$$
  $$(FP.max(y) \leftarrow FP.max(x) + \Delta(x))$$
— règle d'Inconsistance :
  — **R5** : $FP.min(x) > FP.max(x) \models$ REJET

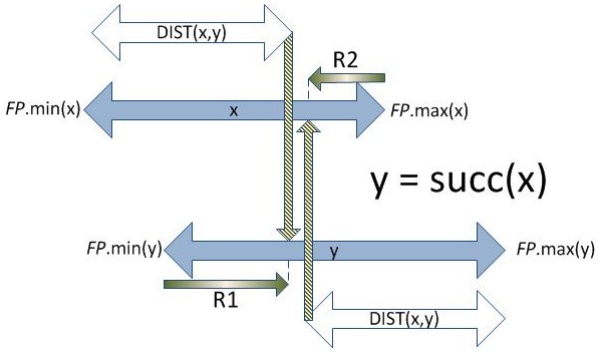


FIGURE 3.5 – Propagation de R1 et R2 sur deux nœuds successifs

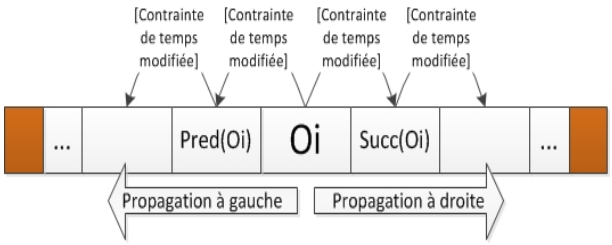


FIGURE 3.6 – Propagation de contraintes pour tester l'insertion d'un nouveau nœud *origine* $o_i$

La propagation de contraintes se fait par l'application de ces 5 règles au sein de l'algorithme PropagerFenetreTemps :

**Algorithme 6** PropagerFenetreTemps

**Entrées:** la tournée $\Gamma_k$, la demande à insérer $D_i$, les fenêtres de temps $FP$ propagées depuis la dernière insertion effective : ($\Gamma_k$,$D_i$,$FP$) ;

**Sorties:** le booléen *booleenTempsValide* indiquant si l'insertion de $D_i$ ne modifie pas la validité de la tournée $\Gamma_k$ et une copie des fenêtres de temps nouvellement propagées : (*booleenTempsValide*,$FP$) ;

1: $\mathcal{L} \leftarrow \{o_i, d_i\}$ ; $booleenTempsValide = VRAI$ ;
2: **Tant que** ($\mathcal{L}$ non vide) et (*booleenTempsValide*) **Faire**
3: $z \leftarrow Tete(\mathcal{L})$ ; $\mathcal{L} \leftarrow ListeSansTete(\mathcal{L})$ ;
4: **Pour** j de 1 à 5 **Faire**
5: chercher toutes les paires (x,y) pour lesquelles peut intervenir la règle $R_j$ et où $z$ correspond au nœud $x$ **ou** au nœud y ;
6: **Pour tout** (x,y) **Faire**
7: appliquer la règle $R_j$ ;
8: **Si** le nœud dont la fenêtre a été modifiée n'est pas dans $\mathcal{L}$ **Alors**
9: insérer le nœud dans $\mathcal{L}$ ;
10: **Fin si**
11: **Si** REJET **Alors**
12: $booleenTempsValide = FAUX$ ;
13: **Fin si**
14: **Fin pour**
15: **Fin pour**
16: **Fin tant que**
17: **Retourner** (*booleenTempsValide*,$FP$) ;

Nous obtenons de ce processus la réduction de fenêtre de $\Gamma_k$ selon l'insertion de $D_i$ dans $\Gamma_k$, sur les points d'ancrage $x, y$. On vérifie aisément :

**Proposition 1.**
Une tournée est *temps-valide* si et seulement si le booléen *booleenTempsValide* résultant de l'appel de la procédure 6 est égal à VRAI. Dans ce cas, pour tout ensemble de dates de passage temps-valide $t(x)$ associées aux fenêtres contractées $FP(x)$, on a $t(x) \in [F.min(x)\,;F.max(x)]$.

**Preuve.** La partie "seulement si" est triviale. Pour la première partie, il suffit par exemple de poser $\forall x \in X, t(x) = F.min(x)$ pour vérifier la validité de $t$.

**Fin Preuve.**

### 3.2.2 Processus d'horodatage

Nous avons jusque là décrit les différents composants de notre méthode qui permettent d'intégrer une ou des demandes dans une tournée tout en vérifiant la non violation des contraintes. Nous allons maintenant voir comment calculer des dates de passage aux différents points de cette tournée suite au processus de propagation de contraintes de telle sorte que la valeur de la tournée ainsi datée soit la meilleure

possible. Considérons donc une tournée valide $\Gamma_k$, dont les fenêtres de temps $FP(x)$, $x \in \Gamma_k$, sont stables au sens du processus de propagation décrit dans les sections précédentes. Rappelons que nous cherchons à fixer les valeurs $t(x)$, $x \in \Gamma_k$, de façon à minimiser $Eval(k,t) = A.T_{Global}(k,t) + B.T_{Ride}(k,t) + C.T_{Wait}(k,t)$. Nous proposons pour cela deux procédures *EvalTour1* et *EvalTour2*. La première est basée sur un processus de *Bellman*. Suite à la propagation des contraintes, *EvalTour1* va tout d'abord faire sortir le véhicule $k$ au plus tard (borne max de la fenêtre du nœud $DepotD_k$). Ensuite, les dates de passages de $\Gamma_k$ sont fixées au plus tôt de proche en proche.

**Algorithme 7** EvalTour1

**Entrées:** $(\Gamma_k, FP)$ ;
**Sorties:** $t = t(x), x \in \Gamma_k$, et la valeur $Eval(k,t)$ associée ;
1: $t(DepotD_k) = FP.max(DepotD_k)$ ; $x \leftarrow DepotD_k$ ;
2: **Tant que** $x \notin DepotA_k$ **Faire**
3: $y \leftarrow Succ(\Gamma_k, x)$ ; $t(y) \leftarrow Sup(FP.min(y), t(x) + DIST(x,y)$ ; $x \leftarrow y$ ;
4: **Fin tant que**
5: **Retourner** $t$ et en déduire $Eval(k,t)$ ;

**Proposition 2.**

Si B et C sont nuls alors la procédure *EvalTour1* fournit une valeur optimale, c'est à dire un minimum de $Eval(\Gamma, t)$, pour une tournée $\Gamma_k$ d'un véhicule k.

**Preuve.**

Pour $B = C = 0$, minimiser $Eval(k,t)$ équivaut à minimiser le coût global soit $t(Queue(\Gamma_k)) - t(Tete(\Gamma_k))$. La proposition se vérifie en considérant 2 cas :

- **Premier cas** : Il existe au moins un élément $x$ de $\Gamma_k$ tel que $t(x) = FP.min(x)$. Notons alors $x_0$ le premier élément de $\Gamma_k$ tel que :
  - $t(x_0) = FP.min(x_0)$,
  - pour tout $y$ tel que $x_0 \ll^{=}_{\Gamma_k} y \ll_{\Gamma_k} Queue(\Gamma_k)$, nous avons $t(Succ(\Gamma_k, y)) - t(y) = DIST(y, Succ(\Gamma_k, y))$ ; la règle d'inférence **R3** (respectée par $FP$) nous permet de déduire que $t(Queue(\Gamma_k)) = FP.min(Queue(\Gamma_k))$ et donc le résultat puisque $t(Tete(\Gamma_k)) = FP.max(Tete(\Gamma_k))$.
- **Second cas** : Pour tout nœud de $\Gamma_k$, nous avons $t(Succ(\Gamma_k, x)) - t(x) = DIST(x, Succ(\Gamma_k, x))$. Le résultat est alors immédiat.

**Fin Preuve.**

Nous venons de voir comment fixer un rendez-vous le long d'une tournée en ne la parcourant qu'une seule fois. Efficace en terme de temps (et en terme d'objectif si $B = C = 0$), cette méthode ne constitue qu'une approximation si $B$ et $C$ sont non nuls. La seconde méthode, *EvalTour2* (cf. algorithme 8), tend dès lors à affiner le résultat fourni par $EvalTour1$ dans le cas où B et C sont non nuls. $EvalTour2$ opère, à partir du système de dates $t = (t(x), x \in \Gamma_k)$ calculé par $EvalTour1$, en effectuant des perturbations améliorantes sur le système $t$. Ces perturbations

consistent à modifier une valeur $t(x)$ particulière, en la faisant monter ou descendre suivant l'impact d'un tel mouvement sur la quantité $Eval(k,t)$. Plus spécifiquement, nous notons que la quantité $Eval(k,t)$ est linéaire en $t(x)$, pour chaque nœud $x$ dans $\Gamma_k$. Notons $\lambda_x$ le coefficient qui accompagne $t(x)$ dans $Eval(k,t)$. Le calcul explicite de $\lambda_x, x \in X$, est facile à effectuer :

— si $Statut(x) = DepotD$ alors $\lambda_x = -(A+B+C)$,
— si $Statut(x) = Origine$ alors $\lambda_x = -B$,
— si $Statut(x) = Destination$ alors $\lambda_x = B$,
— si $Statut(x) = DepotA$ alors $\lambda_x = A+B+C$.

Notons (E2) et (E3) les expressions logiques suivantes :

(E2)

$$(\lambda_x < 0) \wedge (Statut(x) \in \{Origine, DepotD\})$$
$$\wedge (t(x) \neq INF(FP.max(x), t(Succ(\Gamma_k, x)) - DIST(x, Succ(\Gamma_k, x)))$$

(E3)

$$(\lambda_x > 0) \wedge (Statut(x) \in \{Destination, DepotA\})$$
$$\wedge (t(x) \neq SUP(FP.min(x), t(Pred(\Gamma_k, x)) + DIST(Pred(\Gamma_k, x), x))$$

La procédure *EvalTour*2 peut alors s'écrire comme suit :

**Algorithme 8** EvalTour2

**Entrées:** $(\Gamma_k, FP)$ ;
**Sorties:** le système de dates $t = t(x), x \in X$, et la valeur $Eval(k,t)$ induite ;

1: $t = EvalTour1(\Gamma_k, FP)$ ; $Continue = VRAI$ ;
2: **Tant que** *Continue* **Faire**
3: chercher $x$ dans $\Gamma_k$ qui satisfait E2 ou (exclusif) E3 ;
4: **Si** la recherche ne donne rien **Alors**
5: $Continue = FAUX$ ;
6: **Sinon**
7: **Si** x satisfait E2 **Alors**
8: $t(x) = INF(FP.max(x), t(Succ(\Gamma_k, x)) - DIST(x, Succ(\Gamma_k, x)))$ ;
9: **Sinon**
10: $t(x) = SUP(FP.min(x), t(Pred(\Gamma_k, x)) + DIST(Pred(\Gamma_k, x), x))$ ;
11: **Fin si**
12: **Fin si**
13: **Fin tant que**
14: **Retourner** $t = (t(x), x \in X)$ et la valeur $Eval(k,t)$ induite ;

Nous noterons par la suite, *ValTour1*$(\Gamma_k)$ et *ValTour2*$(\Gamma_k)$ les deux mesures de la performance d'une tournée $\Gamma_k$ obtenues respectivement par les appels de *EvalTour1* et *EvalTour2*.

### 3.2.3 Algorithme d'insertions successives

**Rappel : Insertion et tests de validité**

Soit $\Gamma_k$ une tournée valide, $D_i = (o_i, d_i, \Delta_i, F(o_i), F(d_i), Q_i)$ ne figurant pas dans $\Gamma_k$, et 2 nœuds $x$ et $y$, présents dans $\Gamma_k$ de telle sorte que $x \ll^{=}_{\Gamma_k} y$. Nous notons $Inserted(\Gamma_k, x, y, i)$ la tournée obtenue en insérant $D_i$ selon les nœuds d'ancrage $x$ et $y$, c'est à dire en :

— positionnant $o_i$ entre $x$ et Succ($\Gamma_k$, $x$),

— positionnant $d_i$ entre $y$ et Succ($\Gamma_k$, $y$).

La *Charge validité* de la tournée $Inserted(\Gamma_k, x, y, i)$ est testée selon le mécanisme proposé en 3.2.1 de même que celui de propagation de contraintes qui teste la *Temps validité* de $Inserted(\Gamma_k, x, y, i)$. Toutefois, l'exécution du processus de propagation étant consommatrice de temps, nous la filtrons par 2 tests préalables simples, effectués en cascade :

— **TestUnNoeud** : Ce premier test permet d'établir la faisabilité locale (c'est-à-dire sans mise en œuvre du processus de propagation de contraintes) de l'insertion d'un nœud $u$ entre deux nœuds consécutifs $z$ et $z'$ (soit $z' = Succ(\Gamma_k, z)$) déjà intégrés à $\Gamma_k$. Ce test inclut les contraintes de temps et de charge ;

— **TestDeuxNoeuds** : Ce second test établit la faisabilité locale de l'insertion de l'origine $o_i$ et de la destination $d_i$ d'une demande $D_i$ entre deux nœuds consécutifs de $\Gamma_k$, à la fois du point de vue des contraintes de temps et des contraintes de charge.

**Algorithme 9** TestUnNoeud

**Entrées:** la tournée $\Gamma_k$, deux nœuds consécutifs $z$ et $z'$, un nœud u à tester associé à une charge $Q$, $u$ est doté d'une fenêtre de temps $FP(u)$ : ($\Gamma_k$,$z$,$z'$,$u$,$Q$) ;

**Sorties:** Un booléen de valeur VRAI si le nœud u respecte les conditions nécessaires de faisabilité d'insertion dans $\Gamma_k$, de valeur FAUX sinon ;

1: **Retourner** $(FP.min(z) + DIST(z,u) \leq F.max(u)) \wedge (F.min(u) + DIST(u,z') \leq FP.max(z')) \wedge (FP.min(z) + DIST(z,u) + DIST(u,z') \leq FP.max(z')) \wedge (ChT(\Gamma_k, z) + Q \leq CAP)$ ;

**Algorithme 10** TestDeuxNoeuds

**Entrées:** la tournée $\Gamma_k$, deux nœuds consécutifs $z$ et $z'$, deux nœuds $u$ et $v$ à tester associés à une charge $Q$, $u$ et $v$ sont dotés de fenêtres de temps $FP$ : ($\Gamma_k$,$z$,$z'$,$u$,$v$,$Q$) ;

**Sorties:** Un booléen de valeur VRAI si les nœuds $u$ et $v$ respectent les conditions nécessaires de faisabilité d'insertion dans $\Gamma_k$, de valeur FAUX sinon ;

1: **Retourner** $(FP.min(z) + DIST(z,u) \leq F.max(u)) \wedge (F.min(u) + DIST(u,v) \leq F.max(v)) \wedge (F.min(v) + DIST(v,z') \leq FP.max(z')) \wedge (FP.min(z) + DIST(z,u) + DIST(u,v) \leq F.max(v)) \wedge (FP.min(z) + DIST(z,u) + DIST(u,v) + DIST(v,z') \leq FP.max(z')) \wedge (F.min(u) + DIST(u,v) + DIST(v,z') \leq FP.max(z')) \wedge (ChT(\Gamma_k, z) + Q \leq CAP)$ ;

**Processus d'insertions successives**

Tel que décrit dans l'algorithme 11, TestInsertion, tester l'insertion de la demande $D_i$ dans la tournée $\Gamma_k$ se fait donc par appels en cascade de :

— la procédure TestUnNoeud appliquée à $o_i$, $x$, Succ($\Gamma_k$,$x$) puis à $d_i$, $y$, Succ($\Gamma_k$,$y$), ou, si $x = y$, de la procédure TestDeuxNoeuds, appliquée à $o_i$, $d_i$, $x$, Succ($\Gamma_k$,$x$) ;

— la procédure TestCharge testant la *Charge validité* ;

— la procédure PropagerFenetreTemps testant la *Temps validité*.

L'ensemble du processus d'insertion prend en INPUT l'ensemble $D = (D_i = (o_i,d_i, \Delta_i, F(o_i), F(d_i), Q_i), i \in I)$ de toutes les demandes de service, l'ensemble de nœuds $X$ associés tels que $X = \{DepotD_k, DepotA_k, k = 1..K\} \cup \{o_i, d_i \text{ avec } i \in D_i\}$ et enfin les trois critères A, B et C de la fonction Objectif. Tout au long du processus, nous faisons évoluer les variables principales suivantes :

— l'ensemble I1 qui contient les demandes déjà prises en compte dans les $K$ tournées,

— les tournées $\Gamma = (\Gamma_k, k = 1..K)$, pour lesquelles nous avons, pour chaque nœud $x$ de chaque tournée, la fenêtre de temps $FP(x)$, le chargement global $ChT$ et enfin les évaluations courantes $ValTour1$ et $Valtour2$,

Afin de rendre le processus de gestion des contraintes plus efficace, nous maintenons à jour, un certain nombre d'informations additionnelles :

— $NbCarLibre(i)$ nous donne le nombre de véhicules permettant l'insertion de la demande $D_i$,

— $LibreXY(i)$ rassemble quant à lui tous les 4-uplets ($k$, $x$, $y$, $DiffEval$), avec $x \ll^{=}_{\Gamma_k} y$, avec lesquels la procédure TestInsertion($\Gamma_k$, $x$, $y$, $ValTour2$) nous fournit un résultat positif.

**Algorithme 11** TestInsertion

**Entrées:** la nouvelle tournée à tester, son coût et ses fenêtres : $(\Gamma_k, x, y, i, ValTour2)$ ;

**Sorties:** le booléen $InsertPossible$ de valeur VRAI si l'insertion est possible, de valeur FAUX sinon ainsi que $DiffEval$ donnant l'augmentation du coût de la tournée : ($InsertPossible$, $DiffEval$) ;

1: $Val_{Inserted(\Gamma_k,x,y,i)} \leftarrow \infty$ ;
2: **Si** $x \neq y$ **Alors**
3: $InsertPossible \leftarrow (TestUnNoeud(\Gamma_k, x, Succ(\Gamma_k, x), o_i, Q_i) \wedge (TestUnNoeud(\Gamma_k, y, Succ(\Gamma_k, y), d_i, Q_i)$ ;
4: **Sinon**
5: $InsertPossible \leftarrow TestDeuxNoeuds(\Gamma_k, x, Succ(x), o_i, d_i, Q_i)$ ;
6: **Fin si**
7: **Si** InsertPossible **Alors**
8: **Si** TestCharge($\Gamma_k$,x,y,i) **Alors**
9: $(InsertPossible, FP_{new}) \leftarrow PropagerFenetreTemps\ (Inserted(\Gamma_k, x, y, D_i), D_i, FP)$ ;
10: **Si** InsertPossible **Alors**
11: $Val_{Inserted(\Gamma_k,x,y,i)} \leftarrow EvalTour1(Inserted(\Gamma_k, x, y, D_i), FP_{new})$ ;
12: **Fin si**
13: **Fin si**
14: **Fin si**
15: **Retourner** (InsertPossible,$Val_{Inserted(\Gamma_k,x,y,i)} - ValTour2$) ;

L'algorithme général d'insertions successives fonctionne alors comme suit :

1. Le processus sélectionne aléatoirement une demande $i_0$ qui n'a pas encore été insérée dans les différentes tournées. Cette sélection se fait parmi les demandes les plus difficiles à satisfaire, c'est-à-dire celles induisant une petite valeur $NbCarLibre$. (E4)
2. La sélection du véhicule $k$ et des points d'ancrage $x, y \in \Gamma_k$ qui vont permettre l'insertion de $i_0$ est réalisée grâce à deux constantes entières et paramètres de la routine d'insertion : N1 et N2. Nous avons vu que l'évaluation d'un triplet $(x, y, k)$ candidat à l'insertion de $i_0$ pouvait dériver de l'application de la procédure $EvalTour1$, rapide, fournissant $DiffEval$, ou de celle de la procédure $EvalTour2$, plus lente, mais aussi plus précise. Dans un premier temps, les N1 4-uplets $(k, x, y, DiffEval)$ de $LibreXY(i_0)$ ayant les plus petites évaluations $DiffEval$ sont sélectionnées et $EvalTour2$ est alors appliquée à ces N1 4-uplets. On obtient de ces derniers N2 meilleurs 4-uplets triés et stockés dans une nouvelle liste $LCandidats$. Le 4-uplet $(k, x, y, DiffEval)$ donnant lieu à l'insertion de $i_0$ est enfin sélectionné de manière aléatoire parmi ces N2 éléments. (E5)
3. $D_{i_0}$ est alors insérée dans la tournée $\Gamma_{k_0}$ entre les nœuds $x_0$ et $y_0$, la tournée du véhicule $k_0$ devient : $Inserted(\Gamma_{k_0}, x_0, y_0, i_0)$.
4. Les ensembles $LibreXY$ et $NbCarLibre$ sont mis à jour en tenant compte de l'insertion de $i_0$ dans la tournée $\Gamma_k$.

Le processus global d'insertions successives est donné en page suivante par l'algorithme 12.

**Algorithme 12** INSERTION

**Entrées:** deux entiers N1 et N2 (décrits en (E5)), l'ensemble des demandes D, le 4-uplet ($X$, DIST, $K$, $CAP$) et enfin les trois paramètres positifs de la fonction Objectif A, B et C : (N1, N2, $D$, $X$, DIST, $K$, $CAP$, A, B, C) ;

**Sorties:** l'ensemble des tournées $\Gamma$, $k$ de 1 à $K$, l'ensemble des temps de passage $t$, l'évaluation $Eval(\Gamma, t)$ et enfin l'ensemble $Rejet$ des demandes qui ne peuvent être satisfaites : ($\Gamma$, $t$, $Eval(\Gamma, t)$, $Rejet$) ;

1: **Pour tout** $k \in K$ **Faire**
2: $\Gamma_k \leftarrow \{DepotD, DepotA\}$ ; $t(DepotD) \leftarrow 0$ ; $t(DepotA) \leftarrow 0$ ; $I1 \leftarrow Nil$ ; $J \leftarrow I$ ; $Rejet \leftarrow Nil$ ;
3: **Fin pour**
4: **Pour tout** $i \in J$ **Faire**
5: $NbCarLibre(i) \leftarrow K$ ; $LibreXY(i) \leftarrow$ { l'ensemble des 4-uplets ($k$, $x$, $y$, $v$), $k$ de 1 à $K$, $x$ et $y = DepotD_k$, $x <<_{\Gamma_k} y$ et enfin $v$ le coût donné par *EvalTour2* sur l'insertion de $D_i$ } ;
6: **Fin pour**
7: **Tant que** $J \neq Nil$ **Faire**
8: sélectionner une demande $i_0$ selon (E4) et la retirer $i_0$ de l'ensemble $J$ ;
9: **Si** $LibreXY(i_0) = Nil$ **Alors**
10: $Rejet \leftarrow Rejet \cup \{i_0\}$ ;
11: **Sinon**
12: générer *LCandidats* en suivant (E5) et sélectionner un 4-uplet $(k_0, x_0, y_0, v_0)$ toujours comme indiqué en (E5) ; $\Gamma_{k_0} \leftarrow Inserted(\Gamma_{k_0}, x_0, y_0, i_0)$ ; insérer $i_0$ dans $I_1$ ;
13: **Pour tout** $i \in J$ **Faire**
14: rassembler dans une liste $\Lambda(i)$ l'ensemble des paires ($x$,$y$) avec $x <<^{=}_{Inserted(\Gamma_{k_0}, x_0, y_0, i_0)} y$ et de telle sorte qu'il existe un 4-uplet $(k_0, x', y', v)$ dans $LibreXY(i)$ qui satisfait les deux conditions suivantes :
15: 1 - $(x' = x)$ ou $(x' = x_0$ et $x' = Pred(\Gamma_{k_0}, x))$ ou $((x' = x_0 = y_0)$ et $(x' = Pred(Pred(\Gamma_{k_0}, x))$ ;
16: 2 - $(y' = y)$ ou $(y' = y_0$ et $y' = Pred(\Gamma_{k_0}, y'))$ ou $((y' = x_0 = y_0)$ et $(y' = Pred(Pred(\Gamma_{k_0}, y))$ ;
17: **Pour tout** ($x$, $y$) de $\Lambda(i)$ **Faire**
18: $(InsertPossible, ValTour1(\Gamma_k)) \leftarrow TestInsertion(\Gamma_{k_0}, x, y, i)$ ;
19: retirer $(k_0, x, y, v)$ de $LibreXY(i)$ si un tel 4-uplet existe puis décrémenter $NbCarLibre(i)$ si $k_0$ ne figure plus dans $LibreXY(i)$ ;
20: **Si** $InsertPossible$ **Alors**
21: insérer $(k_0, x, y, ValTour1(\Gamma_k))$ dans $LibreXY(i)$ puis incrémenter $NbCarLibre(i)$ si $k_0$ ne figure pas encore dans $LibreXY(i)$ ;
22: **Fin si**
23: **Fin pour**
24: **Fin pour**
25: **Fin si**
26: **Fin tant que**
27: récupérer l'ensemble des dates $t$ après l'avoir calculé avec $EvalTour2$ ;
28: **Retourner** ($\Gamma$, $t$, $Eval(\Gamma, t)$, $Rejet$) ;

**Intégration de l'algorithme INSERTION dans un processus de Monte-Carlo**

L'algorithme INSERTION est non déterministe, et cela du fait de :
- la sélection de la demande $i_0$ qui est réalisée *aléatoirement* sur les N demandes au plus faible nombre de véhicules disponibles ;
- la sélection des paramètres pour insérer la demande $i_0$ ordonnés selon $DiffEval$, c'est à dire de la variation de qualité de la tournée avant et après l'insertion.

Il est donc possible de réaliser, en séquentiel ou parallèle, plusieurs exécutions d'INSERTION de façon à obtenir plusieurs solutions différentes, et à conserver la meilleure. Ceci se fait selon le schéma ci-dessous :

**Algorithme 13** INSERTIONAleatoire

**Entrées:** N, N1 et N2 définis en (E5) ;
**Sorties:** la planification des tournées de véhicules sélectionnée ;
1: **Pour** p de 1 à $P$ **Faire**
2: exécuter la procédure INSERTION ;
3: garder l'ensemble des résultats où le nombre de rejets est le plus petit ;
4: **Fin pour**
5: **Retourner** la solution parmi celles au plus petit nombre de rejets qui a la meilleure performance ;

### 3.2.4 Renforcement du processus d'insertion

Contrairement à ce qui se passe ou sein d'une exploitation réelle, nous exigeons, dans notre modèle du DARP, l'intégration de toutes les demandes qui sont émises. Dans certains cas, notamment lorsque les contraintes de temps sont particulièrement serrées, il peut apparaître que certaines demandes n'ont pu être insérées au terme de la procédure d'insertion alors qu'il existe bien une solution où toutes peuvent l'être. Rappelons que le processus d'insertion comporte deux niveaux de décision : la sélection de la demande et le choix des paramètres pour l'insérer. C'est au travers de ces deux niveaux qu'est impacté le succès (ou l'échec) de l'insertion d'une demande. Nous allons voir comment les renforcer l'un et l'autre.

**Apprentissage simple**

Le premier niveau de décision de notre méthode concerne le choix de la demande à insérer. Il est réalisé selon le nombre de véhicules pouvant encore l'accepter. Cette information peut ne pas être suffisante.

On peut l'affiner en échangeant des informations entre les différentes réplications afin de repérer les différentes demandes qui ont du mal à s'intégrer aux tournées. Il est donc possible d'introduire une notion d'**apprentissage** dans l'algorithme INSERTIONAleatoire, c'est-à-dire au sein du processus de Monte-Carlo. Pour cela, on pondère les éléments de $D$ au niveau de la procédure de sélection de la demande à insérer (E4). Au terme de chaque réplication, le processus repère les demandes rejetées et leur impose un poids fonction du nombre de fois où elles ont été exclues :

la priorité devient plus forte à chaque fois qu'il y a rejet. En effet, plus le poids devient élevé plus la demande peut-être considérée comme critique et difficile à insérer. Ainsi, le processus de sélection sélectionne en priorité les demandes au plus grand poids. Cet apprentissage permet donc de cibler les demandes qui sont difficiles à insérer afin de les sélectionner plus tôt dans le processus général.

**Graphe des demandes non compatibles**

Une autre approche, propre à faciliter la gestion des contraintes serrées, vise à exploiter les contraintes d'exclusion mutuelles portant sur les demandes.

**Coloration d'un graphe de demandes non compatibles.** Soient $D_i$ et $D_j$ deux demandes dans $D$. Nous pouvons préalablement tester les 6 combinaisons de tournées, initialement vides, que peuvent induire ces deux demandes :

— $\Gamma_1^{i,j} = \{DepotD, o_i, d_i, o_j, d_j, DepotA\}$,
— $\Gamma_2^{i,j} = \{DepotD, o_i, o_j, d_i, d_j, DepotA\}$,
— $\Gamma_3^{i,j} = \{DepotD, o_i, o_j, d_j, d_i, DepotA\}$,
— $\Gamma_4^{i,j} = \{DepotD, o_j, d_j, o_i, d_i, DepotA\}$,
— $\Gamma_5^{i,j} = \{DepotD, o_j, o_i, d_j, d_i, DepotA\}$,
— $\Gamma_6^{i,j} = \{DepotD, o_j, o_i, d_i, d_j, DepotA\}$.

Dans le cas où aucune de ces tournées n'est valide, nous disons que $D_i$ et $D_j$ sont non compatibles et nous le notons *NonCompatible(i,j)*. Nous notons également *GrapheNonCompatible = (D,NonCompatible)* le graphe des demandes ainsi induit par la relation *NonCompatible*. Toute solution réalisable d'une instance DARP induit alors une coloration de graphe à $K$ couleurs.

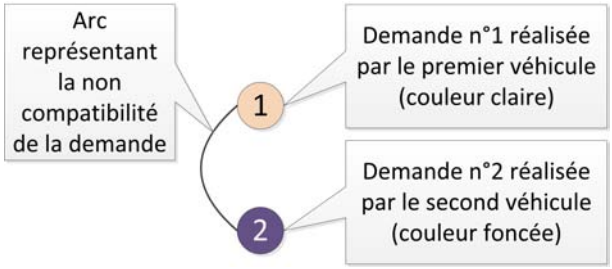


FIGURE 3.7 – *GrapheNonCompatible* avec deux véhicules et deux demandes

Supposons maintenant que les tournées $\Gamma_1..\Gamma_K$ ont été partiellement construites. Ceci induit une coloration partielle de *GrapheNonCompatible*. Il est alors possible :

— d'appliquer à cette coloration partielle des règles de propagation permettant de l'enrichir ou de déduire une impossibilité ;
— de tester l'existence d'une coloration de Graphe *NonCompatible*, qui prolonge cette coloration partielle, et déduire, soit une impossibilité, soit une préconisation sur la demande à insérer.

L'existence d'une telle coloration peut être testée par une simple recherche arborescente. Coloration, l'algorithme 14, effectue cette recherche à partir de l'ensemble H des demandes non traitées, et de la collection $(Dom = Dom(i), i \in H)$ qui fournit,

pour chaque demande $i$ de H, l'ensemble des couleurs (véhicules) encore possibles. Coloration nous renvoie tout d'abord un booléen nous informant de la réussite ou de l'échec de la recherche, une fonction colorante $\gamma(i)$ avec $i \in H$, si elle existe, et une liste de clauses contenant des expressions logiques exprimant des contraintes additionnelles sur les colorations possibles. C'est cette partie du résultat qui fournit des indications sur la prochaine insertion à effectuer. Nous ne parcourons pas l'ensemble de l'arbre : ceci prendrait trop de temps, et, de plus, nous devons appeler fréquemment la procédure de coloration. Les règles d'inférence que nous utilisons, lors de la coloration d'un nœud $i_0$ par $k_0$, sont :

— **R1*** : $|Dom_1(i)| = 1$ et $\gamma(i)$ non assignée $\models \gamma(i)$ prend l'unique coloration $k$ et $i$ est insérée dans $J_0$ ;
— **R2*** : $i$, $i' \in H$ et $k \in K$ sont tels que $\gamma(i) = k, k \in Dom_1(i')$ et nous avons $NonCompatible(i, i') \models k$ est enlevé de $Dom_1(i')$ ;
— **R3*** : $i \in H$ et $k \in K$ sont tels que $k \in Dom_1(i)$ et $k$ n'appartient à aucun sous-ensemble $Dom_1(j), j \in H - J_0 \models$ attribuer $k$ à $\gamma(i)$ et insérer $i$ dans $J_0$ ;
— **R4*** : $i \in H$, $Dom_1(i) = NIL \models Succes \leftarrow FAUX$ ;

**Algorithme 14** Coloration

**Entrées:** les demandes H non traitées et chacun de leur domaine réalisable Dom : (H,Dom) ;
**Sorties:** le booléen SuccesColoration, la fonction de coloration $\gamma$, la liste de clauses Cl : (SuccesColoration,$\gamma$,Cl) ;

1: **Si** H=Nil **Alors**
2: **Retourner** (VRAI,Nil,Nil) ;
3: **Sinon**
4: choisir $i_0 \in H$ ; $Continue \leftarrow VRAI$ ; choisir une sous-liste $L \subset Dom(i_0)$ et l'ordonner. On appelle $L$ la liste de balayage ; $ListeInterdits \leftarrow Nil$ ;
5: **Tant que** Continue et L $\neq$ Nil **Faire**
6: $Res \leftarrow FAUX$ ; $k_0 \leftarrow Tete(L)$ ; $L \leftarrow ListeSansTete(L)$ ; $Dom_1 \leftarrow Dom$ ; $J_0 \leftarrow \{i_0\}$ ; $Succes \leftarrow VRAI$ ;
7: appliquer les 4 règles d'inférences R1*,R2*,R3* et R4* ; on récupère la mise à jour de $Dom_1$ ainsi que le booléen Succes dont la valeur a pu être inversée en **R4*** ;
8: **Si** Succes **Alors**
9: $(SuccesColoration_{Aux}, \gamma_{Aux}, Cl_{Aux}) \leftarrow Coloration(H - J_0, Dom_1)$ ;
10: **Si** $SuccesColoration_{Aux}$ **Alors**
11: $Continue \leftarrow FAUX$ ; $succesColoration \leftarrow VRAI$ ; $\gamma \leftarrow \gamma_{Aux} \cup \{\gamma(i_0), \gamma(i_0) = k_0\} \cup \{\gamma(i_R), \gamma(i_R)$ tel qu'il y a affectation de couleur en $i_R$ dans les règles R1* ou R3*$\}$ ; $Cl \leftarrow \{\gamma(i_0) \neq k, k \in ListeInterdits\} \cup \{(\gamma(i_0) = k_0) \Rightarrow c, c \in Cl_{Aux}\}$ ;
12: **Sinon**
13: on insère $k_0$ dans $ListeInterdits$ ;
14: **Fin si**
15: **Sinon**
16: on insère $k_0$ dans $ListeInterdits$ ;
17: **Fin si**
18: **Fin tant que**
19: **Retourner** (Res,$\gamma$,$Cl$) ;
20: **Fin si**

**Explications.** La première règle R1* affecte directement un véhicule à une demande si celle-ci ne dispose plus que d'un seul véhicule pour la prendre en charge. R2* met de façon triviale $Dom(i), i \neq i_0$. Ensuite, R3* affecte un véhicule à une demande si celui-ci n'est présent dans aucun autre ensemble $Dom$. Enfin, la règle R4* informe des cas de non faisabilité de la prise en charge d'une demande avec le graphe courant de non-compatibles.

**Remarques.** *Coloration* est une heuristique, l'instruction en ligne 6 la rend exacte si le domaine réalisable de $i_0$ est entièrement inclus dans la liste. Il faut garder à l'esprit que, dans la plupart des cas, il existe une coloration et que celle-ci peut-être obtenue avec un petit nombre de "retour-arrière" - ang. *backtrack* -. L'instruction (I1) qui indique le choix de la demande "à colorer" applique le *principe de la variable la plus contrainte*. Autrement dit, les demandes qui semblent avoir le plus de difficultés à s'intégrer dans une tournée (i.e. $|Dom(i)|$ est le plus petit) sont sélectionnées en (I1) de manière prioritaire.

Les clauses résultant de coloration sont sauvegardées afin d'être utilisées lors de l'appel suivant de la procédure Coloration, pour identifier les demandes et les véhicules les plus contraints lors du processus d'insertion.

**Exemple.** La figure 3.8 représente un exemple de graphe de non-compatibles. Les couples de demandes (A ;B), (A ;C), (B ;C), (B ;D), (C ;D) ne pouvant pas être pris en charge par le même véhicule sont reliés par une arête. Les domaines réalisables sont donnés dans la figure, trois véhicules sont disponibles : $K = (1\,;2\,;3)$.

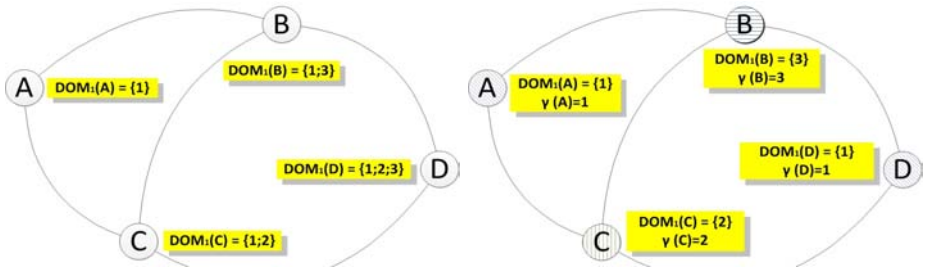


FIGURE 3.8 – Coloration : état initial et final de l'exemple

Toujours pour cet exemple, nous relevons la trace du processus de coloration en $y$ intégrant l'apparition des différentes clauses logiques partant de $A = i_0$ :

1. $\gamma_A \leftarrow 1$ (R1*),
2. $Dom(B) = (3)$ et $Dom(C) = (2)$ ; (R2*),
3. $\gamma_B \leftarrow 3$ puis $Dom(D) = (1;2)$ ; (R1*) puis (R2*),
   (a) une clause est insérée : $\gamma_A = 1 \Rightarrow \gamma_B = 3$ ;
4. $\gamma_C \leftarrow 2$ puis $Dom(D) = (1)$ ; (R1*) puis (R2*),
   (a) une clause est insérée : $\gamma_A = 1 \Rightarrow (\gamma_B \leftarrow 3) \wedge (\gamma_C \leftarrow 2)$ ;
5. $\gamma_D \leftarrow 1$ ; (R1*),
   (a) une clause est insérée : $(\gamma_A = 1 \Rightarrow (\gamma_B = 3) \wedge (\gamma_C = 2)) \Rightarrow \gamma_D \leftarrow 1$.

## 3.3 Résultats expérimentaux

### 3.3.1 Objectifs

Le modèle DARP que nous venons de présenter, qui vise à l'optimisation, au sens multicritère, des 3 critères : temps global, temps d'acheminement et temps d'attente, a été peu étudié. Les différents auteurs tendent en effet à se focaliser sur d'autres critères (cf. [Cordeau and Laporte (2003)], [Cordeau (2006)], [Jorgensen et al. (2007)] et [Parragh et al. (2010)]). Les tests que nous avons menés à bien et dont nous présentons ici les résultats, visent donc ici à évaluer le comportement comparé des différents modèles, et la façon dont nos algorithmes, conçus en premier lieu pour être rapides, flexibles, et aisément adaptables à des contextes dynamiques, sont ou non à mêmes de répondre à des critères évolutifs et à des modifications dans la structure du modèle. Nous avons pour cela réalisé 3 batteries de tests :

— la première reprend le modèle de [Cordeau and Laporte (2003)], orienté sur la minimisation des distances parcourues, et résolu via des méthodes heuristiques complexes, et compare, d'un point de vue de la qualité, pour différentes valeurs des coefficients multicritères A, B, C, les systèmes de tournées obtenus ;
— les 2 séries de tests suivantes concernent, non pas le modèle décrit jusqu'à présent, mais des variantes de ce modèle obtenues par remplacement de notre critère de base : $A.T_{Global}(k,t) + B.T_{Ride}(k,t) + C.T_{Wait}(k,t)$, par des critères alternatifs issus de la littérature ([Cordeau (2006)] et [Jorgensen et al. (2007)]), c'est donc alors la capacité de notre schéma générique par insertion et propagation de contraintes à s'adapter aisément à des nouveaux objectifs, et à obtenir des résultats satisfaisants que nous testons alors.

### 3.3.2 Environnement et performances

**Mesures de performances.** Notre analyse est ici pour partie orientée vers la comparaison des comportements des différents modèles. Pour autant, elle comporte également un volet "analyse de performance" des algorithmes. Le premier critère de performance porte sur le temps CPU, qui doit être pondéré ici des différences entre les infrastructures informatiques utilisées. Dans la première série de tests, qui met en jeu des instances fortement contraintes, on s'intéresse au taux de réussite de notre algorithme, c'est-à-dire au pourcentage :

(3.19)
$$T_{Succes} = 100.\frac{\text{Nombre de réplications où toutes les demandes ont été insérées}}{\text{Nombre de réplications tentées}}$$

Dans les 2 autres séries de tests, on s'intéresse au critère de précision, c'est-à-dire à la capacité de notre algorithme rapide, adapté à des critères alternatifs à approcher les résultats induits par des algorithmes plus complexes et conçus de façon spécifique pour les critères.

**Méthodes programmées.** Toutes les expérimentations traitées dans ce chapitre se basent sur le développement de l'heuristique INSERTION à base de propagation de contraintes. Les différences qui apparaissent lors de ces trois séries de tests reposent sur le processus de datation des différents rendez-vous pris par les véhicules avec les usagers. La première série de tests utilise les procédures $EvalTour1$ et $EvalTour2$ telles qu'elles ont été définies dans ce chapitre. Les deux autres séries évitent $EvalTour2$, plus chronophage.

**Environnement d'exécution.** Pour l'ensemble de ces tests, le programme a été compilé par la branche *g++* de GCC de la plus vieille version maintenue actuellement : la 4.6.3. Le compilateur a été appelé avec l'option $-o2$ qui optimise globalement les performances en intégrant les instructions disponibles chez notre *CPU* où l'exécution de notre programme s'est déroulée, soit sur un unique *thread* de cet *Amd Phenom* cadencé à 2.5 Ghz aidé d'une mémoire vive de 8Go de *DDR3*. La distribution Linux formant l'environnement d'exécution est une *Gentoo 12,1* pour architecture *amd64*.

### 3.3.3 La 1ère batterie de tests

Ces premiers tests portent sur la résolution des instances de [Cordeau and Laporte (2003)]. Pour chacune, et pour toute demande $i \in D_i$, la difficulté est concentrée sur deux types de contraintes de temps (celles associées aux fenêtres des origines $F(o_i)$ et celles en rapport avec les destinations $F(d_i)$), mais aussi sur la durée maximum d'une tournée $\Delta^k, k = 1..K$, fixée à 480 [1]. Ensuite, l'ensemble des temps de service $\delta$ sont tous équivalents à 10 (nous les avons intégrés aux distances), les temps de connexion maximum $\Delta_i$ tous égaux à 90 et enfin la charge associée à chaque demande $Q_i$ est unitaire pour l'ensemble $D$. Les fenêtres de temps, modélisant donc des contraintes serrées, sont partagées en deux groupes : une demande est composée d'une fenêtre serrée (de quelques minutes) et d'une fenêtre très large. Il y a autant de fenêtres larges que de serrées pour une origine, ceci induisant le même partage pour les destinations. Ces caractéristiques se retrouvent dans la vie de tous les jours par un exemple simple : l'arrivée à 8h00 au travail est fixe (fenêtre de destination serrée pour une arrivée entre 7h45 et 8h00) tout comme le départ à 18h00 (fenêtre de départ serrée de 18h00 à 18h15).

Le tableau 3.3 nous renseigne sur les flottes disponibles pour les 20 instances et le nombre de demandes à satisfaire. Les instances sont nommées en partant de pr01 jusqu'à pr20. Il faut noter que [Cordeau and Laporte (2003)] estiment plus difficiles les 10 premières.

1. La contrainte associée est particulièrement serrée. En effet, nous avons réalisé plusieurs tests (en amont de ceux relevés dans ce document) en y intégrant par erreur un temps service (comme nous devions le faire dans la quasi totalité de la matrice des distances) soit en considérant 490. Les résultats obtenus, notamment sur l'instance la plus difficile, la pr10, étaient bien meilleurs sur l'ensemble des instances et précisément sur le taux de succès.

Cette première série de tests ne résout que les cinq premières instances, les combinaisons de pondérations des critères évaluant la qualité des tournées étant en nombre important (la seconde série considère l'ensemble de ces 20 instances).

| Inst. Ra | pr01 | pr02 | pr03 | pr04 | pr05 | pr06 | pr07 | pr08 | pr09 | pr10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Inst. Rb | pr11 | pr12 | pr13 | pr14 | pr15 | pr16 | pr17 | pr18 | pr19 | pr20 |
| K | 3 | 5 | 7 | 9 | 11 | 13 | 4 | 6 | 8 | 10 |
| $\|D\|$ | 24 | 48 | 72 | 96 | 120 | 144 | 36 | 72 | 108 | 144 |

TABLE 3.3 – Populations de véhicules et de demandes pour les instances traitées

Le tableau 3.4 de la page suivante relève les résultats obtenus avec 9 pondérations différentes pour (A, B, C). Nous retrouvons tout d'abord les $T_{Succes}$ et les temps $CPU$ moyens nécessaires à 100 réplications de notre heuristique sur les 5 premières instances de [Cordeau and Laporte (2003)].

Nous voyons clairement l'impact des triplets (A, B, C) sur les distances parcourues par les véhicules. (A=1, B=0, C=-1) est le meilleur triplet si l'objectif est de minimiser les distances sans modifier notre modèle.

[Cordeau and Laporte (2003)] ne cherchant pas à optimiser la qualité de service, le rapport des temps d'acheminement $T_{Ride_C}$ sur ceux obtenus par INSERTION ($T_{Ride}$) lorsque $B = 1$, varie d'un facteur 2 à 3. Ceci signifie que les usagers restent 2 à 3 fois plus de temps dans les véhicules.

La comparaison des temps d'attente $T_{Wait}$ et $T_{Wait_C}$ reflète également la différence des critères optimisés. Les temps d'attente $T_{Wait}$ sont réduits dès lors que A=1 et/ou C=1. Ce qui est naturel pour les résultats avec C=1 l'est aussi pour A=1 du fait que les temps d'attente sont intégrés aux durées des tournées.

Les résultats sont obtenus au bout de seulement 100 réplications d'INSERTION, d'où des temps CPU très bas si on les compare à ceux de [Cordeau and Laporte (2003)]. Un service de transport à la demande se devant d'être réactif, ces temps sont comparables aux temps disponibles dans une exploitation réelle.

| Inst. | A | B | C | $T_{Global}$ | $T_{Ride}$ | $T_{Wait}$ | $T_{Succes}$ | $Dist$ | $CPU$ | $T_{Global_C}$ | $T_{Ride_C}$ | $T_{Wait_C}$ | $Dist_C$ | $CPU_C$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pr01 | 1 | 0 | 0 | 708 | 1030 | 0 | 100 | 228 | 0.2 | 881 | 1095 | 211 | 190 | 1140 |
| | 0 | 1 | 0 | 924 | 396 | 126 | 100 | 318 | | | | | | |
| | 0 | 0 | 1 | 910 | 798 | 95 | 19 | 335 | | | | | | |
| | 1 | 1 | 0 | 878 | 553 | 71 | 98 | 327 | | | | | | |
| | 0 | 1 | 1 | 878 | 553 | 71 | 98 | 327 | | | | | | |
| | 1 | 0 | 1 | 774 | 907 | 23 | 72 | 272 | | | | | | |
| | 1 | 1 | 1 | 878 | 553 | 71 | 98 | 327 | | | | | | |
| | 10 | 1 | 1 | 724 | 884 | 1 | 100 | 243 | | | | | | |
| | 1 | 0 | -1 | 821 | 1384 | 132 | 100 | 210 | | | | | | |
| pr02 | 1 | 0 | 0 | 1515 | 2834 | 72 | 100 | 483 | 0.8 | 1986 | 1977 | 724 | 302 | 4836 |
| | 0 | 1 | 0 | 2097 | 963 | 559 | 98 | 578 | | | | | | |
| | 0 | 0 | 1 | 1819 | 2672 | 294 | 12 | 565 | | | | | | |
| | 1 | 1 | 0 | 1795 | 1255 | 308 | 98 | 527 | | | | | | |
| | 0 | 1 | 1 | 1719 | 1579 | 238 | 97 | 521 | | | | | | |
| | 1 | 0 | 1 | 1638 | 2809 | 184 | 84 | 493 | | | | | | |
| | 1 | 1 | 1 | 1658 | 1657 | 170 | 98 | 528 | | | | | | |
| | 10 | 1 | 1 | 1584 | 2041 | 114 | 98 | 510 | | | | | | |
| | 1 | 0 | -1 | 2076 | 2828 | 764 | 100 | 352 | | | | | | |
| pr03 | 1 | 0 | 0 | 2268 | 3311 | 0 | 99 | 827 | 1.5 | 2579 | 3587 | 607 | 532 | 10308 |
| | 0 | 1 | 0 | 2765 | 1295 | 169 | 99 | 1157 | | | | | | |
| | 0 | 0 | 1 | 2480 | 3199 | 22 | 49 | 1018 | | | | | | |
| | 1 | 1 | 0 | 2529 | 1356 | 55 | 98 | 1034 | | | | | | |
| | 0 | 1 | 1 | 2566 | 1396 | 5 | 99 | 1121 | | | | | | |
| | 1 | 0 | 1 | 2313 | 3722 | 11 | 91 | 862 | | | | | | |
| | 1 | 1 | 1 | 2418 | 1608 | 3 | 99 | 975 | | | | | | |
| | 10 | 1 | 1 | 2332 | 2247 | 4 | 100 | 888 | | | | | | |
| | 1 | 0 | -1 | 2458 | 4061 | 315 | 100 | 703 | | | | | | |
| pr04 | 1 | 0 | 0 | 2828 | 4495 | 1 | 100 | 908 | 2.7 | 3583 | 5021 | 1090 | 573 | 17262 |
| | 0 | 1 | 0 | 3530 | 1621 | 294 | 100 | 1316 | | | | | | |
| | 0 | 0 | 1 | 3213 | 4676 | 30 | 72 | 1264 | | | | | | |
| | 1 | 1 | 0 | 3090 | 1771 | 28 | 100 | 1143 | | | | | | |
| | 0 | 1 | 1 | 3200 | 1715 | 5 | 100 | 1275 | | | | | | |
| | 1 | 0 | 1 | 2890 | 4411 | 0 | 96 | 970 | | | | | | |
| | 1 | 1 | 1 | 3085 | 1837 | 17 | 100 | 1149 | | | | | | |
| | 10 | 1 | 1 | 2836 | 3451 | 4 | 100 | 912 | | | | | | |
| | 1 | 0 | -1 | 3056 | 4973 | 314 | 100 | 822 | | | | | | |
| pr05 | 1 | 0 | 0 | 3439 | 6057 | 18 | 100 | 1021 | 4.4 | 3870 | 6157 | 833 | 637 | 27744 |
| | 0 | 1 | 0 | 4175 | 2000 | 360 | 100 | 1415 | | | | | | |
| | 0 | 0 | 1 | 3801 | 5568 | 19 | 86 | 1382 | | | | | | |
| | 1 | 1 | 0 | 3742 | 2137 | 78 | 100 | 1264 | | | | | | |
| | 0 | 1 | 1 | 3900 | 2151 | 76 | 100 | 1424 | | | | | | |
| | 1 | 0 | 1 | 3385 | 5483 | 0 | 100 | 985 | | | | | | |
| | 1 | 1 | 1 | 3747 | 2240 | 31 | 100 | 1316 | | | | | | |
| | 10 | 1 | 1 | 3408 | 3569 | 2 | 100 | 1006 | | | | | | |
| | 1 | 0 | -1 | 3492 | 6445 | 185 | 100 | 907 | | | | | | |

TABLE 3.4 – Résultats d'$INSERTION$ et de [Cordeau and Laporte (2003)] (suffixe $_C$)

### 3.3.4 La 2ème batterie de tests

La seconde série de tests reprend les instances de la première mais diffère par l'utilisation d'une autre performance : celle introduite par [Jorgensen et al. (2007)] qui comprend 5 critères. L'évaluation d'une tournée de véhicule est composée de :

- la distance parcourue par tous les véhicules *Dist* - ang. : *travel distance* - ;
- le temps de connexion en excès $T_{RideEx}$ - ang. : *excess ride time* - qui retranche les distances données par DIST au temps de connexion $T_{Ride}$. Son calcul est donné par la formule 3.20.
- le temps d'attente passager $T_{WaitPass}$ - ang. : *passenger waiting* -, qui considère le $T_{Wait}$, le volume total de chargement sur un nœud donné $ChT$ et la variation de charge prévue sur celui-ci. Ces trois composants du calcul de $T_{WaitPass}$ permettent de déterminer des temps d'attente proportionnellement au taux de remplissage des véhicules. Sa formule est donnée en 3.21.
- le temps global $T_{Global}$ identique à celui défini dans ce chapitre,
- le temps concernant la somme des durées relatives aux arrivées prématurées $T_{ArrivEarly}$ - ang. : *early arrival* -. Ce critère cherche à modéliser l'arrivée d'un véhicule avant la borne Min de chaque fenêtre de temps pour chaque demande donnée par les usagers. $T_{ArrivEarly}$ s'obtient par la double somme de la formule 3.22, il est également représenté en figure 3.9 de la page suivante par la durée "Arrivée Prématurée".

$$T_{RideEx}(k,t) = \sum_{i \in D_{\Gamma_k}} ((t(d_i) - t(o_i)) - DIST(o_i, d_i)) \tag{3.20}$$

$$T_{WaitPass}(k,t) = \sum_{x=succ(Tete(\Gamma_k))}^{x=pred(Queue(\Gamma_k))} T_{Wait}(x)(ChT(\Gamma_k, x) - Q_x) \tag{3.21}$$

(3.22)

$$T_{ArrivEarly} = \sum_{x=Tete(\Gamma_k)}^{x=pred(pred(Queue(\Gamma_k)))} (F.Min(succ(x)) - (t(x) + DIST(x, succ(x))))$$

$T_{RideEx}$ nous donne $T_{Ride}$ diminué des distances fournies par la matrice DIST. Si la connexion est directe et le rendez-vous sans temps d'attente, $T_{RideEx}$ est nul pour la demande traitée. Il faut cependant garder en tête qu'inclus dans une performance multicritère, $T_{RideEx}$ a beaucoup moins de poids que $T_{Ride}$ puisque sa valeur est bien plus petite. Développons justement la pondération de ces 5 critères. *Dist*, $T_{RideEx}$, $T_{WaitPass}$ , $T_{Global}$ et $T_{ArrivEarly}$ ont respectivement des poids de 8, 3, 1, 1 et $|D|$. Nous pouvons dès lors remarquer que $T_{ArrivEarly}$ est le principal critère sachant que $|D| \in [24, 144]$. L'Objectif utilisé est donné par la formule 3.23.

$$\begin{aligned} Eval(\Gamma, t) = \sum_{k \in K} (&8.Dist_{\Gamma_k}(t) + 3.T_{Ride}(k,t) \\ &+T_{WaitPass}(k,t) + T_{Global}(k,t) + |D|.T_{ArrivEarly}(k,t)) \end{aligned} \tag{3.23}$$

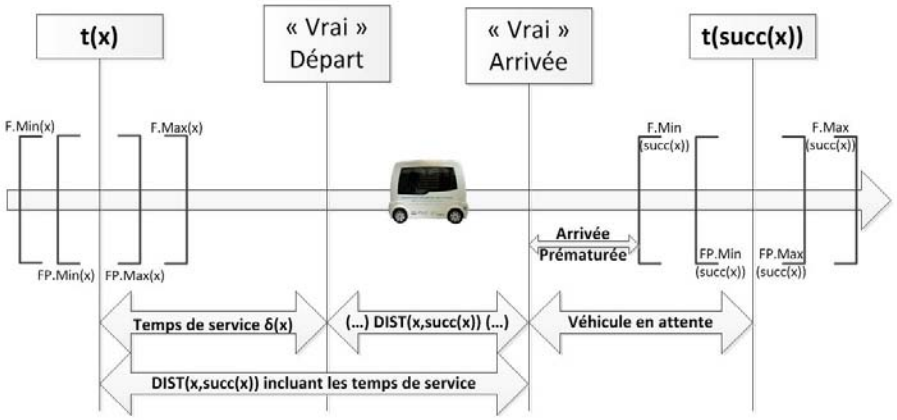


FIGURE 3.9 – Moments entre deux nœuds d'une tournée

La recherche à voisinage variable de [Parragh et al. (2010)] obtient les résultats relevés dans le tableau 3.5.

| Inst. | $Dist$ | $T_{RideEx}$ | $T_{WaitPass}$ | $T_{Global}$ | $T_{ArrivEarly}$ |
|---|---|---|---|---|---|
| pr01 | 273,7 | 49,2 | 0,0 | 863,7 | 1,4 |
| pr02 | 431,2 | 113,5 | 0,3 | 1774,2 | 189,1 |
| pr03 | 777,9 | 322,0 | 0,4 | 2425,9 | 88,2 |
| pr05 | 1023,8 | 246,8 | 0,2 | 3712,2 | 93,4 |
| pr09 | 1045,3 | 529,1 | 12,9 | 3232,2 | 5,7 |
| pr10 | 1389,7 | 594,6 | 4,0 | 4499,5 | 52,9 |
| pr11 | 238,5 | 60,1 | 0,0 | 737,3 | 0,0 |
| pr12 | 431,2 | 25,3 | 0,0 | 1403,7 | 1,5 |
| pr15 | 951,7 | 197,2 | 0,3 | 3417,6 | 6,1 |
| pr16 | 1241,2 | 275,7 | 0,0 | 4183,0 | 10,8 |
| pr17 | 359,2 | 103,3 | 0,0 | 1123,6 | 8,2 |
| pr19 | 977,6 | 338,0 | 0,0 | 3211,7 | 12,7 |
| pr20 | 1310,0 | 493,3 | 1,7 | 4218,4 | 1,7 |
| Moy. | 803,9 | 257,5 | 1,5 | 2677,1 | 36,3 |

TABLE 3.5 – Résultats obtenus par [Parragh et al. (2010)]

Les résultats obtenus par INSERTION au travers de cette seconde série de tests sont relevés dans le tableau 3.6. Nous comparons ensuite en tableau 3.7 l'ensemble de ces résultats ainsi que ceux de [Jorgensen et al. (2007)]. Les différents éléments du tableau sont :

— AG : la performance obtenue par l'algorithme génétique de [Jorgensen et al. (2007)],
— VNS : la performance obtenue par la recherche à voisinage variable de [Parragh et al. (2010)],
— INS : la performance obtenue par notre heuristique INSERTION.

| Inst. | *Dist* | $T_{RideEx}$ | $T_{WaitPass}$ | $T_{Global}$ | $T_{ArrivEarly}$ |
|---|---|---|---|---|---|
| pr01 | 272,8 | 145,6 | 0,0 | 752,3 | 0,0 |
| pr02 | 495,3 | 711,2 | 46,8 | 1625,7 | 8,0 |
| pr03 | 861,8 | 388,6 | 0,0 | 2301,8 | 0,0 |
| pr05 | 1054,6 | 705,2 | 0,0 | 3454,6 | 0,0 |
| pr09 | 1056,2 | 805,2 | 0,0 | 3216,2 | 0,0 |
| pr10 | 1517,7 | 1568,7 | 85,7 | 4553,2 | 155,6 |
| pr11 | 235,8 | 69,2 | 0,0 | 715,8 | 0,0 |
| pr12 | 449,3 | 21,0 | 0,0 | 1409,3 | 0,0 |
| pr15 | 1001,2 | 372,7 | 0,0 | 3401,2 | 0,0 |
| pr16 | 1321,2 | 411,4 | 0,0 | 4201,2 | 0,0 |
| pr17 | 396,0 | 65,4 | 0,0 | 1116,0 | 0,0 |
| pr19 | 1062,2 | 622,1 | 0,0 | 3222,2 | 0,0 |
| pr20 | 1411,3 | 655,0 | 0,0 | 4291,3 | 0,0 |
| Moy. | 856,6 | 503,2 | 10,2 | 2635,4 | 12,6 |

TABLE 3.6 – Résultats obtenus par INSERTION

| Inst. | \|D\| | AG | VNS | INS | Gap |
|---|---|---|---|---|---|
| pr01 | 24 | 4696 | 3234.6 | 3371.4 | -4.1 |
| pr02 | 48 | 19426 | 14640.2 | 9025.7 | 62.2 |
| pr03 | 72 | 65306 | 15969.1 | 10780.8 | 48.1 |
| pr05 | 120 | 213420 | 23852.0 | 14054.2 | 69.7 |
| pr09 | 108 | 333283 | 13806.4 | 14175.7 | -2.6 |
| pr10 | 144 | 740890 | 25016.5 | 35359.5 | -29.3 |
| pr11 | 24 | 4762 | 2825.5 | 2927.6 | -3.5 |
| pr12 | 48 | 13580 | 5003.1 | 5066.5 | -1.3 |
| pr15 | 120 | 98111 | 12360.5 | 12528.9 | -1.3 |
| pr16 | 144 | 185169 | 16499.4 | 16339.4 | 1.0 |
| pr17 | 36 | 9169 | 4601.7 | 4523.1 | 1.7 |
| pr19 | 108 | 167709 | 13412.8 | 13564.9 | -1.1 |
| pr20 | 144 | 474758 | 16420.0 | 17546.5 | -6.4 |
| Moy. | | 179252.2 | 12895.5 | 12251.1 | 5.3 |

TABLE 3.7 – Comparaison des performances obtenues sur les trois approches

Nous voyons à travers le tableau 3.7 que nous avons obtenu des résultats de qualité équivalente à [Parragh et al. (2010)] qui en proposaient déjà de bien meilleurs que [Jorgensen et al. (2007)] et c'est pourquoi le tableau 3.5 ne reprend que les valeurs des 5 critères des meilleures solutions trouvées dans la littérature, il est alors comparable au tableau 3.6 correspondant à l'exécution de notre heuristique. Dans le tableau 3.7, le gap en pourcentage entre l'approche de [Parragh et al. (2010)] (VNS) et nos algorithmes (INS) est donné par Gap.

La comparaison entre les tableaux 3.5 et 3.6 montre quelques différences entre les deux approches : la recherche à voisinage variable - ang. *VNS* - et notre solution. Ces relevés nous montrent que la recherche à voisinage variable est plus performante sur les trois premiers critères et notre approche sur les deux derniers. Notre procédure

*EvalTour1* tente de comprimer l'ensemble d'une tournée $\Gamma_k$ donnée, il n'y a donc pas de surprise à être performant sur $T_{Global}$.

Le troisième et dernier tableau exprime la performance globale des trois approches. En moyenne nous obtenons des solutions bien plus performantes que l'algorithme génétique et pratiquement équivalentes (sinon meilleures) à celles de la recherche à voisinage variable.

L'instance pr10 pose problème puisque nous avons utilisé notre heuristique sous sa forme la plus dépouillée (e.g. sans pondération effective sur l'ordre d'insertion des demandes suivant les rejets d'insertion précédents ou encore, par exemple, l'intégration et l'exécution de la métaheuristique *Backtrack* au moment du rejet) et peu de réplications nous donnaient une solution réalisable. La performance associée étant largement plus grande que les autres, elle fait défaut malgré une meilleure moyenne. En réalité, notre meilleure solution sur pr10 nous fournit un $T_{ArrivEarly}$ qui n'est pas assez minimisé, et, si nous remplaçons sa valeur par celle obtenue par [Parragh et al. (2010)], nous obtiendrions une performance proche de ce dernier au lieu de 35359.5.

### 3.3.5 La 3ème batterie de tests

D'autres instances existent dans la littérature du DARP notamment celles réalisées par [Cordeau (2006)]. La génération de ces nouvelles instances reprend les bases des anciennes ([Cordeau and Laporte (2003)]) : les demandes sont aléatoirement générées sur un carré de côté 20, le temps de parcours entre deux nœuds est la distance euclidienne les séparant et chaque demande a une première fenêtre de temps large et une seconde serrée de 15 minutes. Les 24 instances sont partagées en deux ensembles. Chaque charge des demandes du premier ensemble est unitaire, la capacité est de 3 et $\Delta_i = 30, \forall i \in D_1$, contrairement au second où chaque charge est tirée aléatoirement entre 1 et $CAP$, $CAP = 3$ et $\Delta_i = 45, \forall i \in D_2$. Le tableau 3.8 résume le reste de la volumétrie de ces instances.

| Inst. a | Inst. b | $K$ | $\|D\|$ | Inst. a | Inst. b | $K$ | $\|D\|$ |
|---|---|---|---|---|---|---|---|
| a2-16 | b2-16 | 2 | 16 | a3-36 | b3-36 | 3 | 36 |
| a2-20 | b2-20 | 2 | 20 | a4-16 | b4-16 | 4 | 16 |
| a2-24 | b2-24 | 2 | 24 | a4-24 | b4-24 | 4 | 24 |
| a3-18 | b3-18 | 3 | 18 | a4-32 | b4-32 | 4 | 32 |
| a3-24 | b3-24 | 3 | 24 | a4-40 | b4-40 | 4 | 40 |
| a3-30 | b3-30 | 3 | 30 | a4-48 | b4-48 | 4 | 48 |

TABLE 3.8 – Volumétrie des instances a et b

Nous cherchons à comparer nos résultats à ceux de [Cordeau (2006)] mais aussi avec ceux de [Parragh (2011)]. Ces derniers cherchent à borner l'optimal et à minimiser exclusivement les distances. Nous reprenons à nouveau cet objectif mono-critère. Les tableaux 3.9 et 3.10 relèvent la performance obtenue (soit la somme des distances parcourues par les véhicules de $VH$) ainsi que les temps nécessaires pour atteindre la meilleure solution.

Nous relevons les éléments suivants :

— Lb et Ub : respectivement borne Min et borne Max,
— Opti : la performance optimale,
— INS : la performance obtenue par INSERTION,
— *Gap* : l'écart en pourcentage entre l'optimal et notre résultat,
— $T_{Wait}$ : temps d'attente obtenu par INSERTION,
— $T_{Ride}$ : temps d'acheminement obtenu par INSERTION,
— CPU* et CPU : les durées d'exécution respectivement du meilleur des travaux ([Cordeau (2006)],[Parragh (2011)]) et d'INSERTION. [Parragh (2011)] n'étudiant que la première série d'instances, le second tableau ne se réfère qu'à [Cordeau (2006)].

| Inst. | Lb | Opti | Ub | CPU*(s) | INS | Gap | $T_{Wait}$ | $T_{Ride}$ | CPU(s) |
|---|---|---|---|---|---|---|---|---|---|
| a2-16 | 294,25 | 294,25 | 294,25 | 1,1 | 294,25 | 0,00 | 387,32 | 344,54 | 0,0 |
| a2-20 | 344,83 | 344,83 | 344,83 | 2,6 | 344,83 | 0,00 | 605,44 | 455,32 | 0,1 |
| a2-24 | 431,12 | 431,12 | 431,12 | 8,5 | 431,12 | 0,00 | 536,79 | 603,31 | 0,3 |
| a3-18 | 300,48 | 300,48 | 300,48 | 4,6 | 300,81 | 0,11 | 196,65 | 419,35 | 0,7 |
| a3-24 | 344,83 | 344,83 | 347,42 | 7,6 | 344,83 | 0,00 | 642,72 | 628,86 | 1,5 |
| a3-30 | 494,85 | 494,85 | 494,85 | 9,8 | 495,26 | 0,08 | 721,21 | 732,86 | 16,3 |
| a3-36 | 583,19 | 583,19 | 584,44 | 105,1 | 589,86 | 1,14 | 868,83 | 903,77 | 13,8 |
| a4-16 | 282,68 | 282,68 | 282,68 | 5,6 | 283,10 | 0,15 | 100,72 | 307,00 | 0,3 |
| a4-24 | 375,02 | 375,02 | 378,13 | 5,6 | 376,21 | 0,32 | 527,81 | 581,60 | 94,0 |
| a4-32 | 485,50 | 485,50 | 487,81 | 30,7 | 487,10 | 0,33 | 593,75 | 796,45 | 29,4 |
| a4-40 | 557,69 | 557,69 | 582,26 | 8328,5 | 565,95 | 1,42 | 1 112,33 | 824,32 | 63,3 |
| a4-48 | 668,82 | NA | 709,47 | 14542,6 | 700,30 | NA | 966,85 | 1 132,92 | 30,8 |

TABLE 3.9 – Résultats sur le groupe d'instances a

| Inst. | Lb | Opti | Ub | CPU*(m) | INS | Gap | $T_{Wait}$ | $T_{Ride}$ | CPU(m) |
|---|---|---|---|---|---|---|---|---|---|
| b2-16 | 309,41 | 309,41 | 309,61 | 0,2 | 309,41 | 0,00 | 386,45 | 448,66 | 0,7 |
| b2-20 | 332,64 | 332,64 | 334,93 | 0,0 | 332,64 | 0,00 | 458,17 | 465,23 | 0,6 |
| b2-24 | 444,71 | 444,71 | 445,11 | 0,1 | 444,71 | 0,00 | 475,88 | 674,12 | 2,6 |
| b3-18 | 301,64 | 301,64 | 301,80 | 0,7 | 301,65 | 0,00 | 278,70 | 479,38 | 0,7 |
| b3-24 | 394,51 | 394,51 | 394,57 | 3,6 | 397,47 | 0,75 | 609,62 | 572,61 | 0,9 |
| b3-30 | 531,44 | 531,44 | 536,04 | 6,8 | 534,23 | 0,52 | 785,71 | 857,28 | 3,5 |
| b3-36 | 603,79 | 603,79 | 611,79 | 62,1 | 603,79 | 0,00 | 919,58 | 942,81 | 3,2 |
| b4-16 | 296,96 | 296,96 | 299,07 | 0,8 | 296,96 | 0,00 | 218,97 | 402,16 | 0,1 |
| b4-24 | 371,41 | 371,41 | 380,27 | 5,9 | 371,41 | 0,00 | 490,06 | 567,75 | 2,8 |
| b4-32 | 494,82 | 494,82 | 500,92 | 176,8 | 506,60 | 2,38 | 921,55 | 749,85 | 1,9 |
| b4-40 | 591,76 | 656,60 | 662,91 | 240,0 | 662,74 | 0,94 | 1 013,20 | 1 021,47 | 3,5 |
| b4-48 | 586,91 | 673,80 | 685,46 | 240,0 | 684,83 | 1,64 | 1 458,76 | 1 262,59 | 5,2 |

TABLE 3.10 – Résultat sur le groupe d'instances b

D'emblée, nous pouvons nous satisfaire de retrouver très souvent les solutions optimales fournies par la littérature ou sinon de nous en être approchés très fortement, le plus gros gap étant de 2,38%. De plus, les temps pour les obtenir sont souvent inférieurs aux travaux de recherches cités, il faut surtout remarquer que les temps

n'explosent pas comme c'est le cas dans la littérature pour les deux dernières instances de chaque set. Remarquons que les temps CPU auraient pu être plus faibles si nous nous étions passés de la phase d'horadatage inutile lorsqu'on ne minimise que les distances. Enfin, nous pouvons conclure qu'il est tout à fait possible d'intégrer la minimisation de la distance parcourue comme un objectif compatible avec notre solution.

## Perspectives et conclusion sur la résolution du DARP

Dans ce chapitre, nous avons mis au point une heuristique à bases d'insertions successives pour résoudre les problèmes de *Dial-a-Ride*. Ces problèmes concentrent leurs difficultés principalement sur les contraintes de temps avec deux fenêtres de temps pour chaque demande, une durée maximum pour son acheminement ainsi qu'une borne sur le temps de chaque tournée. Nous utilisons la propagation de contraintes dès lors qu'une insertion est testée. Au final, les résultats obtenus nous donnent de bonnes performances au travers de plusieurs batteries de tests réalisées à partir des instances de [Cordeau and Laporte (2003)] et de [Cordeau (2006)].

La première batterie a pour but de satisfaire à la fois les usagers et l'exploitant, grâce à l'optimisation selon la fonction Objectif "maison" établie dans le modèle. La seconde utilise la fonction Objectif de [Jorgensen et al. (2007)], obligeant ainsi une modification de notre modèle, ainsi que sa reprise par [Parragh (2011)] et compare leurs travaux. Ceux de ce dernier ont donné de bien meilleurs résultats que ceux du premier, et nous avons obtenu, par rapport à leurs tests relatifs à cette fonction Objectif, des résultats un peu meilleurs que ceux de [Parragh (2011)] (ou tout au moins équivalents à ceux-ci). La dernière série de tests ne cherche à minimiser que les distances. Nous obtenons encore de bons résultats alors que le processus de datation n'est pas pris en compte.

Nos résultats peuvent être confrontés à ceux obtenus par des méthodes de recherche locale. Ces derniers gardent l'avantage d'obtenir des résultats moyens satisfaisants (nous ne gardons que les meilleures réplications). Nous pourrions tout à fait intégrer ces méthodes. Mais, en plus de l'adaptabilité des méthodes d'insertion, la rapidité de notre méthode permet d'envisager le cas dynamique de notre problème. Le volume et la difficulté des instances nous permettent d'envisager de bons résultats dans cet autre contexte.

# Chapitre 4

# Modélisation et résolution du DARP statique avec division de la charge - DARPSL

## Introduction

Le problème de *Dial-a-Ride* avec préemption de chargement (DARPSL - ang. *DARP with split loads* -) reprend le problème standard du DARP, en autorisant, pour chaque demande $D$, une décomposition de la charge $Q$ associée en sous-charges, routées selon des itinéraires distincts. Cette hypothèse correspond à des applications où chaque charge $Q$ décrit un ensemble de personnes et d'objets, composant d'une même structure, mais susceptible d'être dissociées lors d'opérations de transport.

Comme précédemment, on considèrera ici que les coûts de chargement / déchargement - appelés temps de service - sont négligeables ou bien proportionnels à la quantité chargée. Le nombre de divisions n'est pas limité (l'ensemble de la flotte de véhicules peut être utilisé pour répondre à l'intégralité d'une demande donnée).

Le caractère préemptif du traitement du chargement permet donc à une flotte de véhicules de satisfaire les requêtes en se partageant une partie du chargement d'une demande donnée. L'état de l'art du DARPSL (chapitre 2) nous montre que l'impact de cette hypothèse de "préemption de charge" a été peu étudié (voir cependant ([Dror and Trudeau (1989)] et [Archetti et al. (2006)]).

Le modèle ayant été posé de façon formelle, et les algorithmes de la section précédente adaptés à cette hypothèse de "charge préemption", on s'intéressera notamment à la capacité éventuelle de cette hypothèse à induire des gains sur les performances des systèmes de tournées obtenus.

## 4.1 Le modèle DARPSL statique

Cette section décrit le modèle du DARPSL en se basant sur le graphe réduit du chapitre 3. Elle définit de nouvelles variables permettant d'introduire la possible division de chaque charge $Q_i$ ouvrant ainsi la possibilité de satisfaire la demande $i$, $i = 1..|D|$, en plusieurs étapes. Le DARPSL se définit de manière simple en gardant les mêmes hypothèses évoquées dans la définition du DARP et en autorisant donc chaque demande $Q_i$ à être divisée en sous-parties. Ces dernières sont alors transportées d'une origine vers une destination à des moments distincts et/ou par des véhicules différents.

Afin de poser ceci de façon formelle, on nommera "schéma de décomposition" de l'ensemble $D$ des demandes, la décomposition de chaque charge $Q_i$ en une somme $Q_i = Q_{i,1} + .. + Q_{i,j} + .. + Q_{i,n(i)}$, $n(i)$ désignant le nombre de sous-parties du chargement de la demande. Ce schéma de décomposition induit alors l'éclatement des couples de nœuds origine et destination $(o_i,d_i)$ en couples distincts $(o_{i,1},d_{i,1})$, .., $(o_{i,j},d_{i,j})$, .., $(o_{i,n(i)},d_{i,n(i)})$. On déduit alors de ce schéma une instance du DARP standard en considérant que chaque demande $i$ de $D$ donne naissance à $n(i)$ demandes distinctes $(o_{i,j}, d_{i,j}, Q_{i,j}), j = 1..n(i)$, contraintes par les mêmes fenêtres de temps que $i$. Résoudre le DARPSL revient alors à trouver le schéma de décomposition des demandes induisant une instance du DARP et dont la valeur optimale soit la plus petite possible.

### 4.1.1 Réseau réduit virtuel

Rappelons que, dans notre modèle du DARP, nous considérons un graphe réduit avec un ensemble de nœuds $X$ et où ces derniers, même s'ils peuvent être localisés exactement au même endroit, sont tous considérés comme différents.

Ici, le réseau réduit virtuel du DARPSL est défini par un ensemble de nœuds $Z$ dérivé de $X$ où chaque nœud $x$ de $X$ est remplacé par un nombre potentiellement infini de nœuds $(x, s)$, $s$ entier naturel. La décomposition d'une charge $Q_i$ en une somme $Q_i = Q_{i,1} + .. + Q_{i,j} + .. + Q_{i,n(i)}$ induira donc l'activation des nœuds $(o_i, s)$,$(d_i, s)$ du réseau virtuel, $s = 1..n(i)$. Pour toute demande $i$, on a $\sum_{s \text{ tq } (o_i,s)\in\Gamma} Q(o_i, s) = Q_i$. De façon plus naturelle, les statuts des nœuds (*origine*, *destination*, *DepotD* et *DepotA*), la relation *Jumeau*, les fenêtres de temps et la matrice de distances $DIST$, s'étendent aux nœuds de ce réseau virtuel.

**Explications.** Formaliser le DARPSL à l'aide d'un réseau virtuel infini peut paraître lourd, compte tenu du fait qu'il est possible de définir le DARPSL en considérant que l'objet à déterminer est un schéma de décomposition des charges qui minimise la valeur optimale du DARP standard induit. Nous procédons ainsi, car c'est cette représentation du problème qui va épouser l'algorithme par insertion décrit dans la suite. Nous procédons en effet, non pas en nous focalisant sur le schéma de décomposition des charges, mais sur les nœuds du réseau virtuel $Z$, que nous créons et rendons actifs de façon incrémentale, au fil des insertions.

### 4.1.2 Tournées, tournées valides, système de tournées admissible

Une tournée est donc une liste de nœuds de $Z$ telle que :

- *Statut(Tete($\Gamma_k$))* = *DepotD* et *Statut(Queue($\Gamma_k$))* = *DepotA* (contraintes de cycle),
- pour tout nœud $z = (o_i, s)$ (respectivement $(d_i, s)$) de $\Gamma_k$, le nœud Jumeau($z$) est nécessairement dans $\Gamma_k$, et nous avons $z \ll_{\Gamma_k} Jumeau(z)$ (respectivement $Jumeau(z) \ll_{\Gamma_k} z$) ; (contraintes de précédence et de couplage),
- aucun nœud $z$ tel que $z = (x, s) \in Z$ ne peut apparaître à deux reprises dans une tournée $\Gamma_k$. Nous pouvons en déduire qu'un même véhicule ne pourra "revenir sur ses pas" pour prendre de nouveau une partie d'une demande donnée.
- pour tout nœud $z$ dans $\Gamma$, $z$ ni $Tete(\Gamma_k)$ et ni $Queue(\Gamma_k)$ alors $Statut(z) \notin$ Depot.

Une tournée $\Gamma_k$ est dite *datée*, si chaque nœud ($x$,$s$) figurant dans $\Gamma_k$ est accompagnées d'une date $t(x, s)$. Une telle tournée datée $\Gamma_k$ est *temps valide* si pour tout $z \in \Gamma_k$ :

- $t(Succ(\Gamma_k, z)) \geq t(z) + DIST(z, Succ(\Gamma_k, z)), z \neq Queue(\Gamma_k)$ (Contrainte des distances),
- pour tout $i \in D, s \in N$, tels que $(o_i, s)$ et $(d_i, s) \in \Gamma_k$, on a, $t(d_i, s) - t(o_i, s) \leq \Delta_i$ et $t(z) \in F(z)$, $F(z)$ fenêtre de temps du nœud $z$,
- la longueur de $\Gamma_k$ est inférieure ou égale à $\Delta_k$.

La tournée $\Gamma_k$ est dite *chargée* si, pour tout $z$ dans $\Gamma_k$, est associé une charge $Q(z)$. Une telle tournée est alors *charge valide* si :

- $Q(Tete(\Gamma_k)) = Q(Queue(\Gamma_k)) = 0$,
- pour tout nœud $(o_i, s)$ de $\Gamma_k$, $0 \leq Q(o_i, s) = -Q(d_i, s) \leq Q(i)$,
- pour tout $z$ de $\Gamma_k$, $ChT(\Gamma_k, z) = \sum_{y=Tete(\Gamma_k)}^{y=z} Q(z) \leq CAP$.

La tournée est *valide* si elle est *temps valide* et *charge valide*. Un système de tournées $\Gamma = (\Gamma_k, k = 1..K)$ datées par une fonction $t$ et chargées par une fonction $Q$, est dit *admissible* si :

- chaque $\Gamma_k$ est *temps* et *charge valide*,
- si un nœud $(x, s), s > 0$, figure dans une des tournées $\Gamma_k, k = 1..K$, du système alors il en est de même pour tout nœud $(x, s'), s' < s$,
- pour toute demande $i \in D, Q_i = \sum_s Q(o_i, s)$.

### 4.1.3 Evaluation d'une tournée

Le modèle DARPSL évalue une tournée $k$ selon trois critères de temps : la durée globale d'une tournée $T_{Global}(k, t)$, la somme des temps nécessaires à la réalisation des demandes pour $\Gamma_k$ donné associée au véhicule $k$ de $K$ soit $T_{Ride}(k, t)$ et les temps d'attente $T_{Wait}(k, t)$ pour les usagers sur cette tournée, leurs formules sont données par les formules 4.1, 4.2 et 4.3. La préemption de charge implique une modification de l'évaluation des tournées. En effet, si la totalité du chargement d'une demande

est acheminée par plusieurs véhicules, les calculs des temps de connexion doivent être redéfinis. Obtenir $T_{Ride}(k,t)$ consiste à calculer pour chacune d'entre elles la différence entre la dernière date d'arrivée et la première date de départ puis de sommer l'ensemble des ces différences. On obtient alors :

$$T_{Global}(k,t) = t(Queue(\Gamma_k)) - t(Tete(\Gamma_k)) \tag{4.1}$$

$$T_{Ride}(k,t) = \sum_{i \in D}((Max_{k \in K \text{ tq } i \in \Gamma_k}(t(d_i))) - (Min_{k \in K \text{ tq } i \in \Gamma_k}(t(o_i)))) \tag{4.2}$$

$$T_{Wait}(k,t) = T_{Global}(k,t) - \sum_{z \in \Gamma - \{DepotD\}} DIST(Pred(z), z) \tag{4.3}$$

A, B et C sont identiques au cas général, ils sont les paramètres évaluant une tournée $\Gamma_k$ par $Eval_{\Gamma_k}(t)$. $Eval(\Gamma, t)$ est la somme des $Eval_{\Gamma_k}(t)$ des $K$ tournées qui nous permet d'évaluer le planning général de la flotte VH, sa formule est donnée en 4.4.

$$Eval(\Gamma,t) = \sum_{k \in K}(A.T_{Global}(k,t) + B.T_{Ride}(k,t) + C.T_{Wait}(k,t)) \tag{4.4}$$

### 4.1.4 Synthèse du modèle DARPSL

Les **INPUT** du DARPSL sont les mêmes que dans le cas standard, c'est-à-dire :

- l'ensemble de demandes $D$ à 6 composantes {l'origine $o_i$, la destination $d_i$, la durée maximum d'acheminement (ou temps de connexion) $\Delta_i$, les deux fenêtres de temps $F(o_i)$ et $F(d_i)$ et enfin la charge $Q_i$} ;
- l'ensemble des nœuds $X$,
- la matrice $DIST$ des distances entre les différents nœuds de $X$,
- le nombre de véhicules $K$ et leur capacité commune $CAP$,
- les trois coefficients positifs A, B et C pondérant la fonction Objectif.

L'objet **OUTPUT** à construire est :

- le système de tournées $\Gamma = (\Gamma_k, k = 1..K)$, définies sur $Z$, datées par une fonction date $t$ et chargées par une fonction charge $Q$.

Au niveau des **Contraintes** : $\Gamma$, le système de tournées, datées par $t$ et chargées par $Q$, doit être admissible au sens défini précédemment. L'**Objectif** est que la quantité $Eval(\Gamma, t)$ soit la plus petite possible.

### 4.1.5 Exemple

Nous allons voir, au moyen d'un exemple simple, que la préemption de chargement peut engendrer un moindre coût si on le compare à celui obtenu à partir de la résolution du problème standard.

Prenons un ensemble de 3 demandes à satisfaire strictement identiques impliquant des contraintes équivalentes en temps et en espace. Aucun temps de service n'est considéré dans cet exemple. Les paramètres nécessaires à l'exemple sont fournis par le tableau 4.1. L'optimisation ne cherche qu'à minimiser les distances parcourues par l'ensemble de la flotte.

| $Q_i$ | $CAP$ | $\|D\|$ | $K$ |
|---|---|---|---|
| 2 | 3 | 3 | 3 |

TABLE 4.1 – Exemple sur la préemption de charge : les paramètres

Pour cette instance, nous voyons clairement que la résolution du DARPSL permet d'obtenir une solution plus performante que celle qui nous est donnée par la résolution du problème standard. En effet, la minimisation de la distance parcourue amènera à ne se servir que de deux véhicules pour satisfaire l'ensemble des demandes. La troisième demande, qui implique la sortie d'un troisième véhicule si le fractionnement est non autorisé, sera divisée en parts égales et permettra aux deux premiers véhicules de se remplir au maximum. Le gain de performance est ici d'environ 33% si l'on s'en tient aux distances et aux trois demandes seules. La figure 4.1 schématise un autre exemple de la prise en charge d'une demande $i, i \in D$, par deux véhicules.

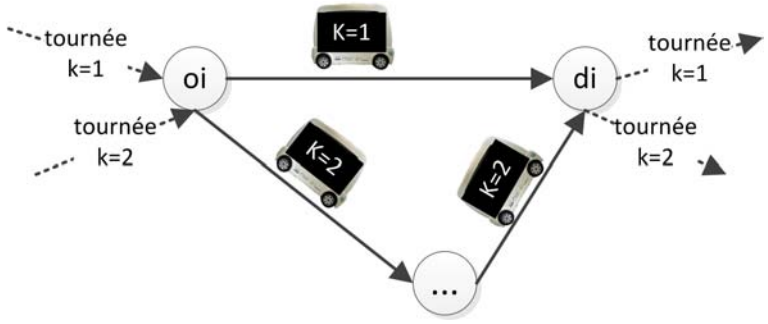


FIGURE 4.1 – Principe de la préemption de charge

Pour une demande donnée $i$, regardons maintenant de plus près comment schématiser le respect des contraintes relatives aux temps de connexion qui sont associés aux deux dates bornant les $2s$ temps de passage en $o_i$ et $d_i$ des $s$ véhicules acheminant la demande. Les fenêtres de temps doivent également toujours borner chaque date de passage (cf. figure 4.2 où les rendez-vous fixés, pour deux tournées prenant un charge une même demande, respectent les contraintes de temps). Pour ce qui est de la charge, la somme de chaque partie d'une demande $i$ doit former $Q_i$. La figure 4.3 représente cette contrainte pour la satisfaction d'une demande $i$ par deux véhicules $k = 1$ et $k = 2$ transportant chacun une sous-partie de $Q_i$.

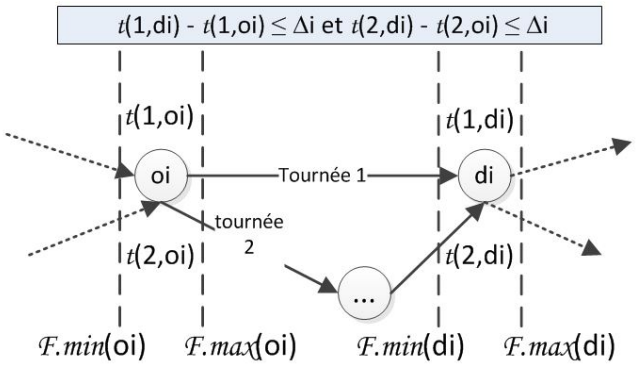


FIGURE 4.2 – Respect des contraintes de temps avec une demande satisfaite par deux tournées

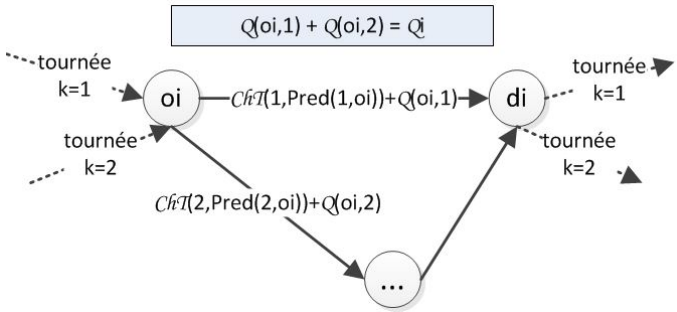


FIGURE 4.3 – Partage d'une charge $Q_i$ sur deux véhicules

## 4.2 Procédés de résolution

Les algorithmes que nous allons présenter constituent une adaptation de nos techniques d'insertion du DARP standard (chapitre précédent). Le processus principal est gardé mais des différences substantielles sont tout de même à noter :

— à chaque insertion, une partie du chargement de la demande traitée, d'indice $i_0$, peut déjà avoir été insérée. Nous noterons $Q_{Aux_{i_0}}$ la charge résiduelle qui reste à planifier pour la demande $i_0$ de chargement globale $Q_{i_0}$. La nouvelle insertion donne naissance à deux nœuds supplémentaires $(o_{i_0}, s)$ et $(d_{i_0}, s)$ avec $1 \leq s' < s$ où $s'$ sont les indices associés aux traitements déjà effectués de la demande $i_0$. La principale difficulté est de calculer la part de charge à acheminer dans la présente insertion. Remarquons que si cette quantité est égale à $Q_{Aux_{i_0}}$ nous avons $s = n(i_0)$ ;

— par conséquent, nous transformons les 4-uplets $(k, x, y, v)$ de *LibreXY* d'une demande $i$ en 5-uplets $(k, x, y, v, q)$ où $q$ est tel que $q \leq Q_{Aux_i}$ et représente la charge maximale pouvant être prise en charge, dès lors qu'une partie de la demande est traitée par le véhicule $k$, selon les points d'ancrage $x$, $y$. La tournée résultant du traitement par le véhicule $k$, selon $x$ et $y$, de cette charge maximale $q$ est désormais notée $Inserted(\Gamma_k, x, y, i, q)$.

Nous pouvons maintenant écrire le principe de l'algorithme résolvant le DARPSL.

### 4.2.1 Principe général

Le principe du processus général de résolution est résumé par l'algorithme 15.

**Algorithme 15** Principe des insertions partielles indépendantes

1: initialiser toutes les tournées et tous les $LibreXY$ ; $Z \leftarrow NIL$ ; $J \leftarrow I$ ; $Rejet \leftarrow NIL$ ;
2: **Pour tout** $i \in J$ **Faire**
3: $Q_{Aux_i} \leftarrow Q_i$ ;
4: **Fin pour**
5: **Tant que** $J \neq Nil$ **Faire**
6: sélectionner une demande $i_0$ ;
7: **Si** $LibreXY(i_0) = Nil$ **Alors**
8: $Rejet \leftarrow Rejet \cup \{i_0, Q_{Aux_{i0}}\}$ ;
9: **Sinon**
10: calculer $s_0$, créer les nœuds $(o_{i_0}, s_0)$ et $(d_{i_0}, s_0)$ puis choisir les paramètres d'insertion ($k_0$,$x_0$,$y_0$,$q_0$) faisant un compromis entre la quantité transportée et l'évolution de la performance des tournées ;
11: compléter les tournées de $k_0$ tel que $\Gamma_{k_0} \leftarrow Inserted(\Gamma_{k_0}, x_0, y_0, i_0, q_0)$ ;
12: mettre à jour les éléments de $LibreXY$ liés à $k_0$ ;
13: **Si** $q_0 = Q_{Aux_{i0}}$ **Alors**
14: retirer $i_0$ de J ;
15: **Sinon**
16: $Q_{Aux_{i0}} = Q_{Aux_{i0}} - q_0$ ;
17: **Fin si**
18: **Fin si**
19: **Fin tant que**

Le processus sélectionne un 5-uplet ($k_0$, $x_0$, $y_0$, $q_0$) relatif à une demande cible $i_0$, insère la charge $q_0$ de $i_0$ dans le véhicule $k_0$ selon les points d'ancrage $x_0$, $y_0$, et réalise une nouvelle itération de la boucle principale. L'itération suivante peut traiter d'une autre demande même s'il reste une quantité non nulle à transporter pour la précédente. Ce processus doit être à présent précisé de façon à obtenir un compromis entre le chargement $q_0$ qui doit être le plus grand possible et la performance $v_0$ induite. La seule recherche d'une meilleure performance indépendamment de la quantité de la demande à transporter risque d'induire à la satisfaction d'une toute petite partie du chargement. Nous cherchons donc une décomposition de la charge permettant la distribution aux tournées de la façon la plus efficace possible. Ceci nous amène à définir, une demande $i_0$ à insérer étant à priori ciblée, le problème de *répartition de la charge* entre les différents véhicules :

**{Problème de distribution de charge :** sélectionner une collection $\Lambda = (k_1, x_1, y_1, v_1, q_1), ...,(k_s, x_s, y_s, v_s, q_s)$ parmi $LCandidats$ de telle sorte que :

- un même véhicule $k$ n'est impliqué que dans au plus un quintuplé de $\Lambda$,
- $\sum_{j=1..s} q_j \geq Q_{i_0}$,
- $\sum_{j=1..s} v_j$ soit la plus petite somme possible.**}**

Ce problème peut être formulé comme celui du sac à dos - ang. *Knapsack Problem* -. Sa NP-Complétude nous a amené à le traiter de façon approchée, via une heuristique, *ChargeDistribution*.

Son principe de cette heuristique très simple : nous limitons la taille de la plus petite charge transportable (limitant ainsi le choix des paramètres d'insertion) selon la capacité des véhicules et la taille moyenne des chargements. Les ensembles $LibreXY$ sont donc tronqués des éléments où $q_0$ est trop petit. Les éléments de $LibreXY$ sont ensuite triés selon le meilleur rapport coût / quantité du chargement et seule une partie de ceux qui répondent le mieux à ce critère formeront l'ensemble $LCandidats$. C'est dans ce dernier petit ensemble que les différentes combinaisons sont testées pour retenir la meilleure. La taille de $LCandidats$ est fixée entre le minimum d'éléments nécessaires à la satisfaction de la demande et une limite qui est paramètre du calcul du nombre de combinaisons permises.

*ChargeDistribution* va modifier la logique de l'algorithme 15 puisque l'ensemble de la demande est traitée à chaque itération (et non une sous-partie du chargement). Nous nous basons maintenant sur le processus général INSERTIONDARPSL dont les grandes lignes sont décrites en algorithme 16. Si cette procédure ne renvoie pas de solution, la demande est rejetée, sinon, $s$ paires de nœuds sont créées et insérées dans leur tournée respective.

**Algorithme 16** Principe de INSERTIONDARPSL

initialiser toutes les tournées et tous les $LibreXY$ ; $Z \leftarrow NIL$ ; $J \leftarrow I$ ; $Rejet \leftarrow NIL$ ;
**Tant que** $J \neq Nil$ **Faire**
  sélectionner une demande $i_0$ ;
  **Si** $LibreXY(i_0) = Nil$ **Alors**
    $Rejet \leftarrow Rejet \cup \{i_0\}$ ;
  **Sinon**
    lancer *ChargeDistribution* pour sélectionner une collection de paramètres d'insertions $\Lambda = (k_1, x_1, y_1, v_1, q_1), ...,(k_s, x_s, y_s, v_s, q_s)$ ;
    appliquer les opérateurs d'insertion paramétrés par les éléments de $\Lambda$ ;
    mettre à jour les éléments de $LibreXY$ liés aux tournées modifiées ;
  **Fin si**
**Fin tant que**

**Remarques sur INSERTIONDARPSL.** La procédure ne reprend donc pas complètement la philosophie du processus général décrit en algorithme 15. La notion de demande résiduelle disparait, une même demande $i_0$ étant globalement insérée d'un coup, du fait de la procédure *ChargeDistribution*, et distribuée entre les véhicules en cours de la même itération de INSERTIONDARPSL. Ceci a notamment pour conséquence qu'un même véhicule ne puisse contribuer plus d'une fois au traitement d'une même demande, ce qui, dans certains cas, peut induire une limitation. Le lecteur vérifiera cependant qu'il serait possible d'adapter INSERTIONDARPSL de façon à s'affranchir de cette restriction.

**Remarques sur l'adaptation des procédures de tests du DARP Standard.** Il est nécessaire d'adapter les procédures *TestUnNoeud* et *TestDeuxNoeud*. En effet, au moment de leur exécution, on ne dispose pas des charges à acheminer sur les tournées concernées. La procédure *TestCharge* doit elle aussi être adaptée, de façon à renvoyer non pas seulement un booléen, mais la valeur maximale d'une charge $q$ susceptible d'être acheminée sur une tournée $\Gamma_k$, dès lors que l'insertion des origines et destinations associées se fait selon les points d'ancrage $x$ et $y$.

### 4.2.2 Synthèse sur la résolution du DARPSL

Nous avons vu que les entrées de ce problème sont : l'ensemble des demandes $D = (D_i = (o_i, d_i, \Delta_i, F(o_i), F(d_i), Q_i), i \in [1; |D|])$, le 4-uplet $(X, DIST, K, CAP)$ et le triplet (A, B, C). Le processus est une heuristique gloutonne d'insertion basée sur la propagation de contraintes et qui est déjà décrite pour le cas général. La boucle principale insère les demandes une à une, en distribuant à chaque étape la charge associée à ces demandes entre les différentes tournées. A chaque itération, le processus dispose des informations suivantes :

- l'ensemble $I_1 \in I$ des demandes déjà insérées dans la flotte de véhicules,
- l'ensemble $Z_1$ des nœuds créés du réseau virtuel $Z$ et établis lors des insertions des demandes de $I_1$,
- l'ensemble des tournées courantes et leurs caractéristiques (performance, charge courante, fenêtres propagées et les rendez-vous associés),
- la connaissance, pour chaque demande $i$ restant à insérer, des véhicules $k$ susceptibles de permettre l'insertion partielle de $i$, des points d'ancrage $x$, $y$, de la charge maximale $q \leq Q_i$ pouvant être acceptée par $k$, et de la valeur $v$ de la tournée induite. Ces informations sont sauvegardées dans des 5-uplets de la forme $(k, x, y, v, q)$ et dans les ensembles $LibreXY(i)$, $i \in [1; |D|]$.

Le processus d'insertion choisit dans un premier temps la demande à insérer (E1) puis distribue cette demande entre les véhicules pour lesquels une insertion partielle est possible (E2). On peut résumer l'ensemble du processus comme suit :

1. La sélection de la demande $i_0$ est réalisée grâce à l'information $NbCarLibre(i_0)$. Si $NbCarLibre(i) = 1$ pour plusieurs demandes $i$ ($i \in [1; |D|]$), la demande sera sélectionnée par tirage aléatoire, sinon on tire aléatoirement $i_0$ dans un ensemble de demandes pour lesquelles $NbCarLibre(i)$ est faible ; (E1)
2. Une fois la demande cible $i_0$ choisie, le processus trie les éléments de $LibreXY(i_0)$ afin de récupérer les N3 5-uplets ayant la plus petite valeur $v$ donnée par *EvalTour1*. Ces valeurs sont précisées par l'appel d'*EvalTour2* et forment les listes $LCandidats$ triées selon leur plus petite nouvelle valeur $v$ ; (E2)
3. La procédure *ChargeDistribution* est alors appliquée pour extraire un ensemble de 5-uplets $\Lambda = (k_1, x_1, y_1, v_1, q_1), ..., (k_s, x_s, y_s, v_s, q_s)$ découlant de $LCandidats$ qui vont acheminer la totalité de $Q_{i_0}$. Le calcul de $\Lambda$ se fait de telle sorte que :
   - un même véhicule apparait dans au plus un élément de $\Lambda$,

— $\sum_{j=1..s} q_j \geq Q_{i_0}$,
— $\sum_{j=1..s} v_j$ est la plus petite somme possible ;

4. Si l'exécution de *ChargeDistribution* se termine par un échec, la demande est insérée dans la liste *Rejet*, sinon, le processus suivant est appliqué :
   — activation des $2s$ nœuds $(o_i, j)$ et $(d_i, j)$, $j \in [1; s]$, de $Z$,
   — insertion effective des sous-charges $q_j$ dans les tournées $\Gamma_j$ selon les points d'ancrage $x_j$, $y_j$, induits,
   — évaluation des tournées ainsi modifiées,
   — mise à jour des ensembles $LibreXY(i)$, $i = 1..|D|$, et des jeux de paramètres associés, en tenant compte des tournées modifiées.

L'ensemble du processus est écrit dans l'algorithme 17 de la page suivante. Il peut également être rendu non déterministe par les procédés aléatoires de Monte-Carlo. L'algorithme appliqué à la division de chargement ne connaît aucune modification par rapport à celui utilisé pour le DARP standard si ce n'est le formatage des paramètres de sortie de la nouvelle méthode d'insertion.

**Algorithme 17** INSERTIONDARPSL

**Entrées:** (N3, $D$, $X$, $DIST$, $K$, $CAP$, A, B, C) ;

**Sorties:** ($\Gamma$, $t$, $Q$, $Eval(\Gamma, t)$, $Rejet$) ;

initialiser les tournées $k$ tel que $\Gamma_k \leftarrow \{DepotD, DepotA\}$ ; initialiser $I_1$, $J$ et $Rejet$ comme vides ;
**Pour tout** $i \in J$ **Faire**
  $NbCarLibre(i) \leftarrow K$ ; $LibreXY(i) \leftarrow$ { l'ensemble des 5-uplets ($k$, $x$, $y$, $v$, $Inf(CAP, Q_i)$), $k$ de 1 à $K$, $x$ et $y \in \{DepotD_k, DepotA_k\}$, $x <<_\Gamma y$ et enfin $v$ le coût supputé par *EvalTour2* sur l'insertion de $D_i$ } ;
**Fin pour**
**Tant que** $J \neq Nil$ **Faire**
  sélectionner une demande $i_0$ selon (E1) et la retirer de $J$ ;
  **Si** $LibreXY(i_0) = Nil$ **Alors**
    $Rejet \leftarrow Rejet \cup \{i_0\}$ ;
  **Sinon**
    créer *LCandidats* selon (E2) et appliquer la procédure *ChargeDistribution* pour créer $\Lambda$ ;
    **Si** aucune liste $\Lambda$ n'a pu être créée **Alors**
      $Rejet \leftarrow Rejet \cup \{i_0\}$ ;
    **Sinon**
      soit $\Lambda = \{k_j, x_j, y_j, v_j, q_j\}, j = 1..s$, obtenu par *ChargeDistribution* ;
      activer les nœuds $(o_{i_0}, j)$,$(d_{i_0}, j)$, $j = 1..s$, du réseau virtuel ; insérer $i_0$ dans $I_1$ ;
      **Pour** $j$ de 1 à $s$ **Faire**
        $\Gamma_{k_j} \leftarrow Inserted(\Gamma_{k_j}, x_j, y_j, i_j, q_j)$ ;
        **Pour tout** $i \in J$ **Faire**
          rassembler dans $\Lambda(i)$ l'ensemble des paires $(x, y)$ de telle sorte qu'il existe un 5-uplet $(k_j, x', y', v, q)$ dans $LibreXY(i)$ qui satisfasse les deux conditions suivantes :
          1 - $(x' = x)$ ou $(x' = x_j$ et $x' = Pred(\Gamma_{k_j}, x))$ ou $((x' = x_j = y_j)$ et $(x' = Pred(\Gamma_{k_j}, x))$ ;
          2 - $(y' = y)$ ou $(y' = y_j$ et $y' = Pred(\Gamma_{k_j}, y'))$ ou $((y' = x_j = y_j)$ et $(y' = Pred(\Gamma_{k_j}, y))$ ;
          **Pour tout** $(x, y)$ de $\Lambda(i)$ **Faire**
            $(InsertPossible, ValTour1(\Gamma_{k_j}), q) \leftarrow TestInsertionDARPSL(\Gamma_{k_j}, x, y, i)$ ;
            retirer $(k_j, x, y, v, q)$ de $LibreXY(i)$ si un tel 5-uplet existe puis décrémenter si besoin $NbCarLibre(i)$ ;
            **Si** $InsertPossible$ **Alors**
              insérer $(k_j, x, y, ValTour1(\Gamma_{k_j}), q)$ dans $LibreXY(i)$ puis incrémenter si besoin $NbCarLibre(i)$ ;
            **Fin si**
          **Fin pour**
        **Fin pour**
      **Fin pour**
    **Fin si**
  **Fin si**
**Fin tant que**
calculer $t$ via la procédure $EvalTour2$ (optimisation finale des dates de rendez-vous) ;
**Retourner** ($\Gamma$, $t$, $Q$, $Eval(\Gamma, t)$, $Rejet$) ;

## 4.3 Résultats expérimentaux

### 4.3.1 Objectifs

Les expérimentations sur le DARPSL visent à évaluer l'impact de la préemption de charge sur la faisabilité des instances et la valeur des tournées obtenues par leur résolution. Pour cela, deux séries de tests ont été réalisées. La première se concentre sur les instances de [Cordeau and Laporte (2003)] déjà étudiées dans le chapitre précédent, celles-ci étant modifiées quant aux quantités à transporter pour chaque requête. Le seconde cherche à résoudre des instances "maison" dont la difficulté ne se concentre plus que sur les contraintes de temps mais aussi sur les charges.

### 4.3.2 Protocole expérimental

Pour les méthodes développées ici, l'environnement de développement en C++ et l'exécution sur machine UNIX restent identiques au chapitre précédent.

**Indicateurs et mesure de performance.** Au travers des 3 batteries de tests, nous mémorisons :

- le nombre de véhicules moyen mis en jeu pour chaque demande $T_{Part} = \frac{\sum_{i \in D} n(i)}{|D|}$ ;
- le pourcentage de demandes acceptées $T_{Insert} = \frac{100(|D| - |Rejet|)}{|D|}$.

**Remarque :** Nous considérons ici qu'une demande est satisfaite si la totalité de la charge est acheminée en respectant les contraintes du problème.

**Les gaps relevés.** Nous nous intéressons également aux quantités suivantes, qui fournissent d'autres éléments de comparaison entre les modèles DARP et DARPSL dans l'hypothèse où toutes les demandes sont insérables :

- *GapGlobal* correspond au gap entre les durées des tournées obtenues par résolution du DARP puis du DARPSL. Nous avons $GapGlobal = 100((T_{Global} - T_{Global_{SL}})/T_{Global_{SL}})$ ;
- *GapRide* nous indique le gap entre les deux sommes des temps de connexion, encore appelés *Ride Time*, obtenues par les deux méthodes. Son calcul est formulé par : $GapRide = 100((T_{Ride} - T_{Ride_{SL}})/T_{Ride_{SL}}$ ;
- *GapDist* est le gap sur les distances parcourues par l'ensemble de la flotte (hors temps d'attente) calculées par les deux résolutions. *Dist* est ici la somme des distances parcourues par une collection de routes $\Gamma$. Le gap s'établit par : $GapDist = 100((Dist - Dist_{SL})/Dist_{SL})$ ;
- *GapSucces* donné par la formule : $GapSucces = 100((T_{Succes} - T_{Succes_{SL}})/T_{Succes_{SL}})$ est le gap entre les taux de réussite obtenus dans les deux solutions ;
- *GapInsert* reflète le calcul de *GapSucces* soit : $GapInsert = 100((T_{Insert} - T_{Insert_{SL}})/T_{Insert_{SL}})$. Il nous donne le gap entre les deux taux d'insertion donnés par les deux résolutions.

### 4.3.3 La 1ère batterie de tests

Nous avons dans un premier temps repris les 20 instances (pr01 jusqu'à pr20) de [Cordeau and Laporte (2003)] en ne modifiant que le volume des quantités à transporter (initialement unitaires) et ceci sans changer la capacité des véhicules ($CAP = 6$). Leur difficulté concerne largement les contraintes de temps et non pas le chargement.

Si nous regardons les solutions obtenues dans les résolutions précédentes, la capacité des véhicules est loin d'être totalement utilisée. Nous avons alors étudié le comportement de nos heuristiques en multipliant par $\lambda$ l'ensemble des chargements, $\lambda$ étant constant pour une même batterie. La formule 4.5 donne le calcul des nouvelles charges.

$$q_{i_{modif}} = \lambda q_i, \forall i \in D \tag{4.5}$$

Les valeurs $\lambda$ testées varient ici de 1 à 7. Les instances sont alors traitées une première fois par INSERTION (résolution du DARP) puis par INSERTIONDARPSL (résolution du DARPSL). Chacune de ces résolutions fait l'objet de 50 réplications.

**Remarques :** La capacité des véhicules pour ces instances est de 6 et, bien évidemment, les taux d'*insérabilité* induits par $\lambda = 7$, sans possibilité de préemption sur la charge, sont automatiquement nuls.

Selon la même logique, pour $\lambda = 7$, le taux de partition des demandes $T_{Part}$ obtenu par INSERTIONDARPSL est forcément supérieur ou égal à 2.

Pour les deux résolutions, nous posons ici A=2, B=1, C=0, ce qui revient à dire que la quantité à minimiser s'écrit :

$$Eval(\Gamma, t) = \sum_{k \in K} Eval_{\Gamma_k}(t) = \sum_{k \in K} (2.T_{Global}(k,t) + T_{Ride}(k,t)) \tag{4.6}$$

Les tableaux 4.2 et 4.3 de la page suivante comparent les valeurs $T_{Insert}$ obtenues respectivement par INSERTION et INSERTIONDARPSL.

| Inst. | $\lambda_1$ | $\lambda_2$ | $\lambda_3$ | $\lambda_4$ | $\lambda_5$ | $\lambda_6$ |
|---|---|---|---|---|---|---|
| pr01 | 99.75 | 99.75 | 99.75 | 92.83 | | |
| pr02 | 97.71 | 98.38 | 97.42 | 92.29 | | |
| pr03 | 98.61 | 98.81 | 98.81 | 98.08 | | |
| pr04 | 99.23 | 99.54 | 99.46 | 98.60 | | |
| pr05 | 98.63 | 98.97 | 98.65 | 97.15 | | |
| pr06 | 99.58 | 99.78 | 99.67 | 97.75 | | |
| pr07 | 98.67 | 98.67 | 96.00 | 86.56 | | |
| pr08 | 94.92 | 94.89 | 95.19 | 87.28 | | |
| pr09 | 91.33 | 91.50 | 90.56 | 85.20 | | |
| pr10 | 90.07 | 91.01 | 89.90 | 87.33 | | |
| pr11 | 100.00 | 100.00 | 100.00 | 98.00 | | |
| pr12 | 100.00 | 100.00 | 100.00 | 97.92 | | |
| pr13 | 99.39 | 99.33 | 99.42 | 98.19 | | |
| pr14 | 99.56 | 99.44 | 99.81 | 96.83 | | |
| pr15 | 99.82 | 99.82 | 99.98 | 99.33 | | |
| pr16 | 99.11 | 99.92 | 99.81 | 99.44 | | |
| pr17 | 99.22 | 99.22 | 99.44 | 93.61 | | |
| pr18 | 98.33 | 98.33 | 97.94 | 94.72 | | |
| pr19 | 95.81 | 95.81 | 95.02 | 88.11 | | |
| pr20 | 95.83 | 95.83 | 93.81 | 88.71 | | |
| Moy. | 97.78 | 97.95 | 97.53 | 93.90 | | |

TABLE 4.2 – $T_{Insert}$ de INSERTION

| $\lambda_1$ | $\lambda_2$ | $\lambda_3$ | $\lambda_4$ | $\lambda_5$ | $\lambda_6$ | $\lambda_7$ |
|---|---|---|---|---|---|---|
| 99.92 | 99.67 | 99.92 | 99.75 | 96.17 | 92.67 | 79.83 |
| 98.38 | 98.17 | 96.46 | 94.75 | 90.75 | 93.83 | 77.71 |
| 98.97 | 99.31 | 98.94 | 97.14 | 95.33 | 98.31 | 85.56 |
| 99.77 | 99.56 | 99.25 | 98.83 | 98.19 | 99.04 | 90.23 |
| 98.70 | 99.48 | 99.03 | 98.17 | 95.63 | 95.15 | 87.17 |
| 99.71 | 99.86 | 99.53 | 99.15 | 98.15 | 97.96 | 89.58 |
| 98.83 | 98.56 | 98.22 | 96.22 | 90.11 | 87.00 | 75.83 |
| 95.39 | 95.53 | 93.78 | 91.50 | 85.94 | 87.50 | 72.19 |
| 92.35 | 92.15 | 90.46 | 87.44 | 82.83 | 86.33 | 70.07 |
| 91.50 | 91.46 | 90.24 | 87.67 | 82.44 | 86.90 | 71.19 |
| 100.00 | 100.00 | 100.00 | 100.00 | 99.75 | 97.33 | 93.50 |
| 99.75 | 99.75 | 99.75 | 99.79 | 99.75 | 97.75 | 93.13 |
| 98.97 | 98.97 | 99.33 | 99.53 | 98.83 | 98.08 | 84.50 |
| 99.23 | 99.29 | 99.23 | 99.56 | 99.31 | 97.83 | 89.40 |
| 99.72 | 99.67 | 99.72 | 99.95 | 99.98 | 98.98 | 94.20 |
| 99.78 | 99.99 | 99.83 | 99.82 | 99.60 | 98.92 | 90.94 |
| 99.78 | 99.78 | 99.78 | 99.00 | 97.89 | 96.39 | 83.94 |
| 99.11 | 99.11 | 98.69 | 97.64 | 96.75 | 93.22 | 78.17 |
| 95.11 | 95.09 | 95.41 | 95.15 | 92.26 | 89.70 | 73.09 |
| 95.14 | 95.47 | 93.26 | 93.53 | 89.82 | 88.74 | 72.40 |
| 98.01 | 98.04 | 97.54 | 96.73 | 94.48 | 94.08 | 82.63 |

TABLE 4.3 – $T_{Insert_{SL}}$ de INSERTIONDARPSL

**Commentaires des résultats d'INSERTION.** Les moyennes obtenues, pour un facteur de charge allant du simple au triple ($\lambda = 1..3$), montrent clairement que la contrainte de charge des 20 instances n'est pas du tout serrée. Les $T_{Insert}$ sont encore tout proches du maximum avec $\lambda = 3$ et restent très élevés pour $\lambda = 4..6$. Cela traduit une spécificité de ces instances, qui sont telles que les solutions obtenues induisent une très faible mutualisation de charge, et cela du fait de fenêtres de temps très étroites.

**Commentaires des résultats d'INSERTIONDARPSL.** On voit que la préemption de charge aide à la prise en compte des demandes de façon sensible pour $\lambda = 4..7$. Le tableau 4.4 nous donne les taux de partition moyens $T_{Part}$ pour chacune des valeurs de $\lambda$.

| Inst. | $\lambda_1$ | $\lambda_2$ | $\lambda_3$ | $\lambda_4$ | $\lambda_5$ | $\lambda_6$ | $\lambda_7$ |
|---|---|---|---|---|---|---|---|
| pr01 | 1.00 | 1.00 | 1.00 | 1.17 | 1.19 | 1.00 | 2.00 |
| pr02 | 1.00 | 1.00 | 1.00 | 1.18 | 1.26 | 1.00 | 2.00 |
| pr03 | 1.00 | 1.00 | 1.00 | 1.17 | 1.25 | 1.00 | 2.00 |
| pr04 | 1.00 | 1.00 | 1.00 | 1.14 | 1.24 | 1.00 | 2.00 |
| pr05 | 1.00 | 1.00 | 1.00 | 1.15 | 1.34 | 1.00 | 2.01 |
| pr06 | 1.00 | 1.00 | 1.00 | 1.11 | 1.21 | 1.00 | 2.01 |
| pr07 | 1.00 | 1.00 | 1.00 | 1.21 | 1.25 | 1.00 | 2.00 |
| pr08 | 1.00 | 1.00 | 1.00 | 1.17 | 1.25 | 1.00 | 2.00 |
| pr09 | 1.00 | 1.00 | 1.00 | 1.19 | 1.32 | 1.00 | 2.01 |
| pr10 | 1.00 | 1.00 | 1.00 | 1.23 | 1.42 | 1.00 | 2.01 |
| pr11 | 1.00 | 1.00 | 1.00 | 1.10 | 1.13 | 1.00 | 2.00 |
| pr12 | 1.00 | 1.00 | 1.00 | 1.08 | 1.12 | 1.00 | 2.00 |
| pr13 | 1.00 | 1.00 | 1.00 | 1.06 | 1.08 | 1.00 | 2.00 |
| pr14 | 1.00 | 1.00 | 1.00 | 1.05 | 1.10 | 1.00 | 2.00 |
| pr15 | 1.00 | 1.00 | 1.00 | 1.06 | 1.12 | 1.00 | 2.00 |
| pr16 | 1.00 | 1.00 | 1.00 | 1.04 | 1.09 | 1.00 | 2.00 |
| pr17 | 1.00 | 1.00 | 1.00 | 1.08 | 1.09 | 1.00 | 2.00 |
| pr18 | 1.00 | 1.00 | 1.00 | 1.14 | 1.21 | 1.00 | 2.01 |
| pr19 | 1.00 | 1.00 | 1.00 | 1.13 | 1.23 | 1.00 | 2.01 |
| pr20 | 1.00 | 1.00 | 1.00 | 1.11 | 1.24 | 1.00 | 2.01 |
| pr20 | 1.00 | 1.00 | 1.00 | 1.11 | 1.24 | 1.00 | 2.01 |

TABLE 4.4 – $T_{Part}$ de l'heuristique avec préemption de charge sur instances de [Cordeau and Laporte (2003)] avec multiplication du chargement de 1 à 7

Nous retrouvons le résultat (attendu) de [Nowak et al. (2008)] où est conclu que le gain est maximisé dès lors que le volume requis par chaque demande dépasse la moitié de la capacité des véhicules. L'expérimentation suivante traitera de nouvelles instances générées pour suivre un scénario bien précis. Les demandes de celles-ci requièrent également toutes un même volume mais celui-ci a une parité différente de la capacité des véhicules qui, elle-même, est impaire.

### 4.3.4 La 2ème batterie de tests

Nous avons généré ce paquet d'instances selon un premier scénario bien précis, qui vise à rendre possible, pour chaque instance, la satisfaction de toutes les demandes, et cela selon le mode sans préemption de charge. La figure 4.4 représente ce premier scénario où les lieux des origines des différentes requêtes sont disposés sur une même abscisse "à gauche" et les destinations toutes rassemblées sur un même point "à droite". Les dépôts sont à mi-chemin entre les deux types d'arrêts.

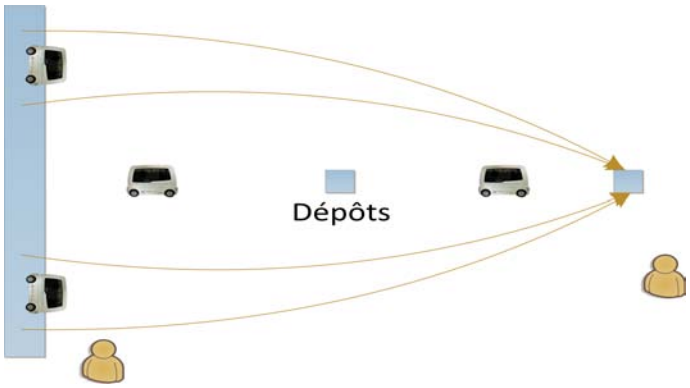


FIGURE 4.4 – DARPSL - Série 2

Les fenêtres de temps sont larges pour les origines et étroites pour les destinations. Le temps de connexion d'une demande $i$ entre ces fenêtres est borné par $\Delta_i$ qui est générée par la somme de la plus longue distance trouvée entre les deux points d'une demande et un nombre aléatoire variant de 0 à l'écart de temps entre $F.max(o_i)$ et $F.min(d_i)$. L'intégration de la plus longue distance n'est pas anodine puisqu'ici nous voulons serrer la contrainte de chargement et donc, pour ces instances, permettre aux véhicules de naviguer dans "l'espace des origines". La capacité des véhicules est fixée à 5 et la charge des différentes demandes à 2. Un véhicule qui utilisera toute sa capacité sera donc un véhicule qui verra l'optimisation de son plan de route utiliser la préemption de chargement. Une fois ces paramètres fixés, nous avons réalisé quatre batteries de 10 instances où chacune se forme par la population de la flotte évoluant de 4 à 10 véhicules et un nombre de demandes variant en proportion de 20 à 65.

Nous recherchons ici optimiser la qualité des tournées par le triplet ($A = 1$, $B = 0$, $C = -1$) donnant la formule d'évaluation 4.7. Comme nous l'avions vu dans les résultats du chapitre précédent, ces paramètres permettent de minimiser indirectement les distances parcourues par les véhicules. Nous testons de façon comparative, INSERTION et INSERTIONDARPSL, en réalisant à chaque fois 50 réplications.

$$Eval(\Gamma, t) = \sum_{k \in K} Eval(k, t) = \sum_{k \in K} (T_{Global}(k, t) - T_{Wait}(k, t)) \tag{4.7}$$

Le tableau 4.5 relève les résultats obtenus par INSERTION, le tableau 4.6 quant à lui relève ceux d'INSERTIONDARPSL et enfin le tableau 4.7 compare les deux techniques. Les critères de temps et distances relevés proviennent des réplications où l'ensemble des demandes a pu être inséré et où ces critères ont été les meilleurs.

| $K$ | $\|D\|$ | $T_{Global}$ | $T_{Ride}$ | $Dist$ | $T_{Succes}$ | $T_{Insert}$ | CPU(s) |
|---|---|---|---|---|---|---|---|
| 4 | 20 | 836.3 | 953.8 | 416.9 | 71.2 | 98.1 | 0.3 |
| 6 | 35 | 1366.7 | 1590.7 | 714.2 | 58.0 | 98.0 | 0.7 |
| 8 | 50 | 1831.0 | 2203.1 | 1038.4 | 28.4 | 96.3 | 1.4 |
| 10 | 65 | 2349.7 | 2758.5 | 1347.2 | 25.2 | 95.8 | 2.4 |
| Moy. | | 1595.9 | 1876.5 | 879.2 | 45.7 | 97.0 | 1.2 |

TABLE 4.5 – Résultats Série 2 - Résolution du DARP

| $K$ | $\|D\|$ | $T_{Global}$ | $T_{Ride}$ | $Dist$ | $T_{Part}$ | $T_{Succes}$ | $T_{Insert_{SL}}$ | CPU(s) |
|---|---|---|---|---|---|---|---|---|
| 4 | 20 | 856.3 | 1113.1 | 356.8 | 1.205 | 93.2 | 99.6 | 0.5 |
| 6 | 35 | 1269.2 | 1787.2 | 605.5 | 1.216 | 78.4 | 98.8 | 1.3 |
| 8 | 50 | 1689.7 | 2542.1 | 850.2 | 1.216 | 53.6 | 97.8 | 2.6 |
| 10 | 65 | 2024.7 | 2981.0 | 1100.3 | 1.220 | 47.6 | 97.1 | 4.2 |
| Moy. | | 1460.0 | 2105.9 | 728.2 | 1.214 | 68.2 | 98.3 | 2.1 |

TABLE 4.6 – Résultats Série 2 - Résolution du DARPSL

| $K$ | $\|D\|$ | GapGlobal | GapRide | GapDist | GapSucces | GapInsert |
|---|---|---|---|---|---|---|
| 4 | 20 | -1.3 | -14.9 | 15.9 | -16.5 | -1.1 |
| 6 | 35 | 7.7 | -11.0 | 18.0 | -25.2 | -0.8 |
| 8 | 50 | 8.6 | -13.0 | 22.2 | -48.5 | -1.5 |
| 10 | 60 | 16.2 | -7.2 | 22.5 | -52.4 | -1.4 |
| Moy. | | 7.8 | -11.5 | 19.6 | -35.6 | -1.2 |

TABLE 4.7 – Résultats Série 2 - Comparaison DARP / DARPSL

**Commentaires.** Le *GapDist* moyen indique une diminution des distances parcourues de 19,6% dès lors que la préemption de charge est autorisée. Ceci s'explique par le fait que la préemption de charge permet de mieux exploiter toute la capacité des véhicules. De la même façon, on constate un gain sensible sur le taux de réplications insérant l'ensemble des demandes et de la même façon sur les pourcentages de demandes insérées par réplication.

## Perspectives et conclusion sur le DARPSL

Une caractéristique essentielle du DARPSL tient dans le fait qu'il est possible de diviser le chargement d'une demande pour l'acheminer grâce à plusieurs véhicules. Ainsi, les modifications du modèle permettent la division de la charge comme le dédoublement des nœuds origine et destination. L'heuristique de propagation de contraintes s'adapte particulièrement bien.

Trois grandes batteries de tests ont été réalisées. Dans la première, les changements apportés aux instances de [Cordeau and Laporte (2003)] (longuement étudiées et mises en œuvre dans le chapitre consacré au DARP) ont permis d'envisager une augmentation de la charge des différentes demandes. Quant aux deux suivantes, elles ont été réalisées grâce à la génération d'instances que nous avons nous-même instaurée. Les trois ont donné de bons résultats, montrant l'intérêt que peut avoir la préemption de chargement dans divers cas. Quand ceux-ci sont spécifiques, c'est peut-être encore plus vrai, comme dans la deuxième batterie de tests ou plus généralement lorsque les nœuds origines et/ou destinations ne sont éloignés ni en espace ni en temps. D'autres tests auraient pu bien évidemment être réalisés puisque le nombre possible de scénarios, dans le cas où la problématique de l'exploitation de la capacité des véhicules est majeure, est très important, mais ces trois batteries doivent sans doute suffire à démontrer les bénéfices d'une telle technique.

Ces tests ont montré que la préemption de chargement pouvait à la fois améliorer le taux d'insertions mais aussi la performance. Une conséquence à tirer des différentes séries de tests est que la division des chargements peut être plus utile dans certains cas que d'autres. Nombreux sont les paramètres à étudier avant de valider la mise en place de cette extension du DARP ; les temps de service, le modèle des flux de demande, le temps à disposition pour l'optimisation et le type de chargement en sont les quatre principaux. D'autres scénarios peuvent faire l'objet d'études plus importantes comme la gestion des camions de déménageurs. Le modèle du DARP avec préemption de charge peut également être intégré dans une chaîne de production à *grande échelle géographique* où les couples de fenêtres de temps définissent des périodes et les durées maximales de connexion décrivant la limite de temps que les produits ne peuvent pas dépasser dans un véhicule et ceci pour diverses raisons (températures, taux d'humidité, etc.).

L'étape suivante sur l'étude du DARPSL consistera à modéliser et optimiser l'arrivée dynamique des demandes qui doivent être traitées par des véhicules déjà en train de satisfaire d'autres requêtes (intégrées aux tournées suite aux calculs de précédentes optimisations). Notre algorithme a été conçu pour que cela se fasse aisément. Il est également possible d'ajouter d'autres fonctionnalités au DARP comme la préemption de véhicules qui transbordent le chargement d'une demande deux à deux dans le but de l'acheminer de manière performante. C'est l'objet du chapitre suivant.

# Chapitre 5

# Modélisation et résolution du DARP statique avec transferts - DARPT

## Introduction

Le *Dial-a-Ride* avec transferts (DARPT) reprend le problème standard (DARP) complété de la possibilité de transborder le chargement d'une demande au cours de son acheminement. Nous parlerons alors de préemption de véhicules. Du fait du contexte réactif dans lequel s'intègre un service de transport à la demande, nous simplifions le problème en imposant que, au plus, un seul transbordement ait lieu lors de la prise en charge d'une demande.

La notion de transbordement utilisée ici est aussi celle de correspondance (voir par exemple, un trajet en train de Lille à Marseille avec changement à Paris). Introduire cette possibilité de préemption de véhicules permet le sectorisation d'une flotte, et donc une réduction de ses coûts.

Nous reprenons dès lors le modèle générique du DARP, et nous l'adoptons à notre hypothèse de préemption de véhicules. Nous résolvons le problème selon le même mode heuristique, par insertions, présenté aux chapitres précédents, et introduisant une continuité entre statique et dynamique.

Dans le chapitre 2, concernant l'état de l'art sur le DARPT, nous avons vu que cette variante du DARP avait été très peu étudiée. On peut se reporter aux travaux sur les problèmes de ramassage et livraison avec transferts - ang. *Pickup and Delivery Problem with Transfers* (PDPT) - (voir par exemple les travaux de [Mitrovic-Minic and Laporte (2006)] et [Masson et al. (2013)] qui résolvent le problème de manière approchée par une recherche locale). Mais, la prise en compte de fenêtres de temps est rare.

Nous continuons sur la même structure que celle des chapitres précédents. Nous décrivons tout d'abord le modèle pour ensuite détailler l'adaptation de notre méthode heuristique au cas avec transferts. Cette dernière est enfin évaluée dans la dernière partie relative aux expérimentations.

## 5.1 Le modèle DARPT statique

LE DARPT reprend le *Dial-a-Ride Problem* standard comme nous l'avons décrit dans les chapitres précédents mais en y intégrant la possibilité suivante : celle de satisfaire une demande en plusieurs fois, à l'aide de plusieurs véhicules et de mécanismes de correspondance. Nous ne parlons plus ici du partage du chargement mais de la subdivision du chemin à parcourir. Une charge à transporter peut, dans sa totalité, être acheminée en plusieurs étapes, dont chacune se rapporte à un unique véhicule $k \in K$ transportant la totalité de $Q_i$, charge de la demande $i \in D$. Il existe quatre types de trajets pour un véhicule prenant en charge la demande $i$ : l'acheminement d'un nœud *origine* $o_i$ à un nœud relais, d'un nœud relais jusqu'à un autre nœud relais (situation exclue de notre modèle), d'un nœud relais jusqu'au nœud de destination $d_i$ ou en une seule fois : de $o_i$ à $d_i$. La figure 5.1 représente le transbordement d'une telle demande $i$ entre deux véhicules. Le véhicule d'index 1 parcourt une première partie du parcours à partir de $o_i$ et transborde la charge sur la tournée 2 sur un point relais. C'est par ce deuxième véhicule que la charge $Q_i$ atteindra la destination $d_i$.

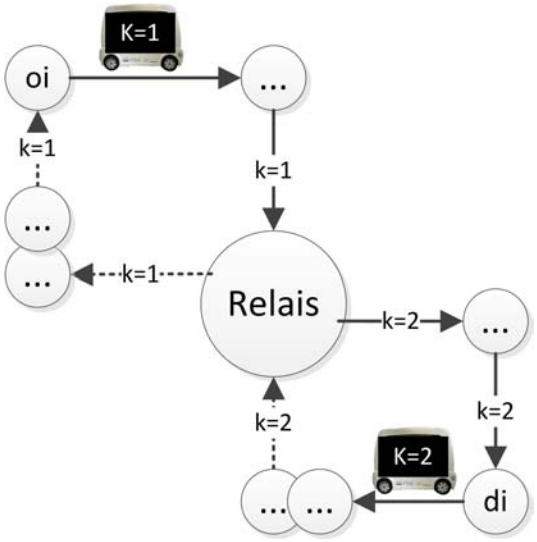


FIGURE 5.1 – Exemple de transbordement d'une demande $i$ par deux véhicules

Les applications pratiques ne peuvent concerner que 2 voire 3 véhicules pour l'acheminement d'un chargement ici indivisible, le transport à la demande n'étant déployé encore aujourd'hui que sur de petites distances. Par souci de simplicité, nous restreindrons donc notre modèle à 2 véhicules par demande, soit 2 étapes maximum.

La principale différence avec le modèle du *Dial-a-Ride* statique standard tient à la gestion de l'ensemble des nœuds $X$. Ce dernier est composé, en plus des dépôts, exclusivement de paires de jumeaux *origine/destination*. Cet ensemble réduit est insuffisant pour traduire l'échange du chargement entre 2 véhicules. Il nous faut pouvoir manipuler des nœuds relais qui permettent cet échange. Dans notre modèle, ces nœuds sont gérés de manière dynamique.

### 5.1.1 Réseau réduit

Nous partons de l'ensemble de nœuds $X = \{DepotA_k, DepotD_k, k = 1..K\} \cup \{o_i, d_i, i \in D\}$ utilisé dans les chapitres précédents. Cet ensemble $X$ est complété d'un ensemble de nœuds relais $U$, que nous allons nous efforcer de gérer de manière incrémentale. L'extension de cet ensemble $X$ par l'ajout de nœuds relais doit être écrite formellement. En entrée du problème, le DARPT intégre les nœuds relais rassemblés dans un ensemble $U$. Ceux-ci, bien que notre modèle se limite à un maximum d'un nœud relais par demande, sont définis de manière implicite du fait que leurs caractéristiques n'ont pas encore été calculées. En effet, c'est lors de la construction des tournées que les nœuds seront clairement définis.

En ce qui concerne les distances séparant chaque couple de nœuds, nous considérons que la fonction $dist()$ a été élargie au calcul de nouvelles distances nées de l'intégration des nœuds relais et ainsi utilisée dans la mise à jour de la matrice $DIST$.

Les données d'entrée du problème DARPT se définissent donc comme suit :
- l'ensemble des demandes $D$, avec pour chaque demande $i, i \in D$ :
  - les fenêtres de temps $F(o_i)$ et $F(d_i)$,
  - la charge $Q_i$,
  - la borne $\Delta_i$ ;
- l'ensemble des $K$ véhicules homogènes $k$ de la flotte $VH$,
- leur capacité $CAP$,
- l'ensemble des nœuds $X = \{DepotD_k, DepotA_k, k = 1..K\}$ U $\{o_i,\ d_i$ avec $i \in D)$,
- l'ensemble des nœuds relais $U$,
- la matrice des distances (sous forme de temps) $DIST$ définie sur $X \cup U$ ;
- les coefficients multicritères A, B, C, permettant de spécifier la mesure de performance.

**Remarque : Représentation implicite de l'ensemble $U$ des nœuds relais.** Afin de limiter les difficultés algorithmiques liées à la taille de l'ensemble $U$ des nœuds relais, celui-ci ne constituera pas véritablement un INPUT pour nos algorithmes du DARPT. En lieu et place, nous utiliserons un générateur ad hoc de nœuds relais, MILIEU, qui, à partir de 2 nœuds actifs $x$, $y$ de $X \cup U$, produira un "milieu" $u$ de $x$, $y$.

Notons $Z$ l'ensemble de nœuds comme l'union de tous les nœuds de $X$ et de tous les nœuds relais de $U$. De fait, les nœuds de $U$ qui nous intéressent seront les nœuds actifs créés par la fonction $MILIEU$. Chacun de ces nœuds apparaît alors exactement dans deux tournées : une première fois en tant qu'émetteur et une seconde en tant que récepteur.

Afin de conserver la logique du réseau virtuel du *Dial-a-Ride* standard, nous distinguons les fonctions récepteur et émetteur par deux nœuds relais distincts. Pour cela, nous associons, à l'ensemble $U$ des nœuds relais créés, l'ensemble des nœuds virtuels $U^*$ suivant :

— $U^* = \{(u, i, -1), (u, i, 1), i \in D, u \in U\}$ ; $(u, i, -1)$ est le nœud émetteur d'où sort la charge $Q_i$ du tour $\Gamma_k, k \in K$, et $(u, i, 1)$ est le nœud récepteur qui permet l'intégration de cette charge dans la tournée $\Gamma_{k'}, k' \in K$. Cela signifie que $Q_i$ est transportée à partir de $o_i$ jusqu'à $u$ par le véhicule $k$ avant que $k'$ n'arrive pour récupérer le chargement en $u$ afin de l'acheminer jusqu'à $d_i$. La figure 5.2 où $k = 1$ et $k' = 2$ schématise l'utilisation de tels nœuds pour une demande $i \in D$.

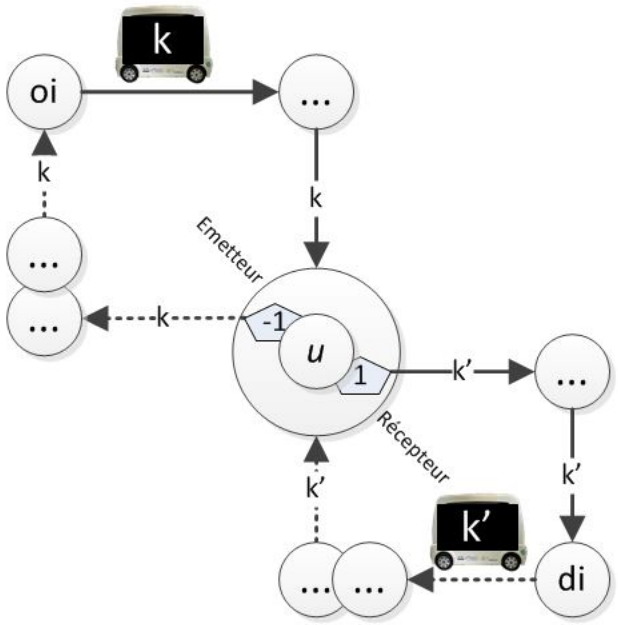


FIGURE 5.2 – Transbordement sur un nœud relais dédoublé en émetteur et récepteur

Nous posons alors $Z^* = X \cup U^*$ et notons $node()$ la fonction polymorphe de $Z^*$ dans $Z$ définie par : pour tout $u \in U$, $node(u, i, -1) = node(u, i, 1) = u$ et, pour tout $x \in X, node(x) = x$. Nous étendons les fonctions *Statut*, *Jumeau*, *Dem*, *CH*, $\Delta$, $F$ à tout nœud $(u, i, \epsilon)$ de $U^*$ :

— $Statut(u, i, -1) = ChargeOut$
— $Statut(u, i, 1) = ChargeIn$,
— $Jumeau(u, i, +1) = (u, i, -1)$
— $Jumeau(u, i, -1) = (u, i, 1)$,
— $Dem(u, i, +1) = Dem(u, i, -1) = i$,
— $CH(u, i, -1) = -Q_i$ et $CH(u, i, +1) = Q_i$,
— $\Delta(u, i, -1) = \Delta(u, i, +1) = +\infty$,
— $F(u, i, -1) = F(u, i, +1) = [0, +\infty[$.

### 5.1.2 Tournées, tournées *valides*, système de tournées *admissible*

Une tournée, au sens du problème de *DARP* avec transferts, est constituée d'une séquence de nœuds notée $\Gamma_k$ qui respecte les propriétés suivantes :

- $Statut(Tete(\Gamma_k)) = DepotD$ ; $Statut(Queue(\Gamma_k)) = DepotA$ ;
- $VI(Tete(\Gamma_k)) = VI(Queue(\Gamma_k)) = k$ ;
- $Statut(x) \notin Depot, \forall x \neq Tete(\Gamma_k)$ et $x \neq Queue(\Gamma_k)$ ;
- aucun nœud $z$ de $Z^*$ n'apparaît plus d'une fois dans la même tournée $\Gamma_k$ ;
- pour tout $i \in D$, une et exactement une seule de ces assertions doit être vérifiée :
  - $o_i$ et $d_i$ apparaissent dans $\Gamma_k$ tels que $o_i \ll_{\Gamma_k} d_i$ et aucun nœud $(u, i, \epsilon), \epsilon \in \{-1, 1\}$ ne s'y trouve ;
  - aucun nœud $o_i$, $d_i$ et $(u, i, \epsilon), \epsilon \in \{-1, 1\}$ ne fait partie de $\Gamma_k$ ;
  - $o_i$ et $(u, i, -1)$ figurent dans $\Gamma_k$ de telle sorte que $o_i \ll_{\Gamma_k} (u, i, -1)$ ;
  - $(u, i, +1)$ et $d_i$ figurent dans $\Gamma_k$ et $(u, i, 1) \ll_{\Gamma_k} d_i$.

La tournée $\Gamma_k$ est alors *charge valide* si et seulement si pour tout $x$ de $\Gamma_k$, $x \neq Tete(\Gamma_k)$, nous avons $\sum_{y=Tete(\Gamma_k)}^{y=x} CH(y) \leq CAP$. La tournée $\Gamma_k$, datée par une fonction date $t$, est *temps valide* si et seulement si pour tout $x$ de $\Gamma_k$ :

- $t(Succ(\Gamma, x)) \geq t(x) + DIST(x, Succ(\Gamma, x)), x \neq Queue(\Gamma)$ ;
- si $Statut(x) \notin \{ChargeOut, ChargeIn\}$, $|((t(Jumeau(x))) - t(x))| \leq \Delta(x)$ ;
- $t(x) \in \mathrm{F(x)}$.

Une tournée $\Gamma_k$, datée par une fonction date $t$, est *valide* si elle est *charge valide* et *temps valide*. Un système de tournées $\Gamma = (\Gamma_k, k = 1..K)$, daté par une fonction $t$, est alors dit *admissible* si :

- chaque tournée $\Gamma_k, k = 1..K$, datée par $t$, est valide, commence en $DepotD_k$ et se termine en $DepotA_k$ ;
- pour chaque demande $i \in D$, on a :
  - les nœuds $o_i$ et $d_i$ apparaissent chacun dans exactement une tournée $\Gamma_k$, et on a : $t(o_i) \leq t(d_i) \leq t(o_i) + \Delta_i$ ;
  - si $o_i$ et $d_i$ apparaissent dans la même tournée $\Gamma_k$, alors aucun nœud de la forme $(u, i, \epsilon)$ n'apparaît dans $\Gamma$ ;
  - si $o_i \in \Gamma_k$ et $d_i \in \Gamma_{k'}$, $k \neq k'$, alors il existe un nœud $(u, i, -1)$ dans $\Gamma_k$ et son jumeau $(u, i, 1)$ dans $\Gamma_{k'}$ et on a : $t(u, i, -1) \leq t(u, i, 1)$ ; aucun autre nœud de la forme $(v, i, \epsilon)$, $v \in U$, $\epsilon \in \{-1, 1\}$, ne figure dans $\Gamma$.

#### Evaluation d'une tournée

Pour tout système $\Gamma = \{\Gamma_{k_1}..\Gamma_K\}$ de tournées *admissible*, associé à un sous-ensemble d'index des demandes $D$ disposant de dates $t$ *temps valide*, on redéfinit, de manière naturelle, les critères de coût du *Dial-a-Ride* standard soit :

- pour chaque tournée $k$ :

$$T_{Global}(k, t) = t(Queue(\Gamma_k)) - t(Tete(\Gamma_k)) \tag{5.1}$$

$$T_{Wait}(k,t) = T_{Global}(k,t) - \sum_{x \in ListeSansTete(\Gamma_k)} DIST(Pred(x), x) \tag{5.2}$$

— on pose alors :

$$Eval(\Gamma, t) = \sum_{k \in K} (A.T_{Global}(k,t) + C.T_{Wait}(k,t)) + B.\sum_{i \in D} (t(d_i) - t(o_i)). \tag{5.3}$$

### 5.1.3 Synthèse des INPUT et OUTPUT du DARPT

Les **INPUT** et **OUTPUT** du DARPT sont établis comme suit :

— Les **INPUT** regroupent **:**
  — une flotte de véhicules $VH$ de population $K$ et de capacité $CAP$,
  — un ensemble $D$ de demandes $D_i, i \in D_i$ à 6 composantes : $o_i$, $d_i$, $\Delta_i$, $F(o_i)$, $F(d_i)$ et $Q_i$,
  — les coefficients positifs A, B, C des trois critères d'appréciation,
  — un ensemble $Z$ qui accroît l'ensemble $X$ par les nœuds relais de $U$ définis de manière implicite,
  — la fonction $Dist$ et son domaine $Z.Z$ permettant ainsi d'élargir la matrice des distances avec les nœuds relais ;

— Les **OUTPUT** rassemblent :
  — un système $\Gamma_k, k = 1..K$, *admissible*, associé aux demandes de $D$ et daté par une fonction date $t$,
  — l'objectif est de faire en sorte que $Eval(\Gamma, t)$ soit le plus petit possible.

## 5.2 Procédés de résolution

### 5.2.1 Principe général

Nous allons à nouveau procéder par insertions successives des demandes de $D$ dans l'ensemble de tournées $\Gamma$ jusqu'à obtenir un système de tournées *admissible*. Nous distinguons deux types de prise en charge d'une demande : une première où la charge est acheminée directement par un unique véhicule et une seconde où l'acheminement se fait en plusieurs étapes, chacune réalisée par un véhicule différent. Ceci nous amène à introduire deux opérateurs d'insertion : $INSERT$ et $INSERTT$.

$INSERT$ est l'opérateur pour le DARP standard. Rappelons que si $k$ est l'index du véhicule, $x$ et $y$ deux nœuds dans la tournée de celui-ci tel que $x \ll^{=}_{\Gamma_k} y$ et $i$, l'index de la tournée, $INSERT(\Gamma_k, x, y, i)$ désigne alors la nouvelle tournée obtenue suite à l'insertion, dans le véhicule $k$, de la demande $i$ tel que l'origine $o_i$ est insérée entre $x$ et $Succ(x, \Gamma_k)$ et la destination $d_i$ entre $y$ et $Succ(y, \Gamma_k)$. $INSERT(\Gamma_k, x, y, i)$ nous fournit donc la tournée $k$ ainsi mise à jour.

$INSERTT$ insère une demande $i$ dans 2 tournées $\Gamma_k$, $\Gamma_{k'}$, $k \neq k'$, en utilisant 5 nœuds $x$, $y$, $x'$, $y'$ et $u \in U$ :

— l'origine $o_i$ est insérée dans $\Gamma_k$ entre $x$ et son successeur $Succ(\Gamma_k, x)$ ;
— la destination $d_i$ est insérée dans $\Gamma_{k'}$ entre $y'$ et son successeur $Succ(\Gamma_{k'}, y')$ ;
— le nœud relais émetteur $(u, i, -1)$ est inséré dans $\Gamma_k$ entre $y$ et $Succ(\Gamma_k, y)$ ;
— le nœud relais récepteur $(u, i, 1)$ est inséré dans $\Gamma_{k'}$ entre $x'$ et $Succ(\Gamma_{k'}, x')$ ;

Cet opérateur $INSERTT$ s'appliquera donc à un jeu de paramètres ($i$, $k$, $k'$, $x$, $y$, $x'$, $y'$, $u$) sous l'écriture $INSERTT(i, k, k', x, y, x', y', u)$. Au cas où son application est faisable, son résultat sera constitué du système de tournées $\Gamma$, modifié selon le processus d'insertion.

Le processus de résolution se décompose alors en deux étapes :

— la première phase opère sur le modèle DARP standard via l'utilisation exclusive de l'opérateur INSERT. Il arrive alors que certaines demandes soient rejetées ;
— la seconde phase rassemble les demandes rejetées et s'efforce de les insérer à l'aide de l'opérateur $INSERTT$.

### 5.2.2 Gestion des paramètres de INSERTT

**Les ensembles $LibreXYT_{o(i)}$ et $LibreXYT_{d(i)}$**

Rappelons que, lors de l'insertion d'une demande dans le processus de résolution du DARP, les ensembles $LibreXY$ recensent, pour chaque demande, l'ensemble des paramètres permettant une application de l'opérateur $INSERT$. Dans le cas du DARPT, nous considérons l'ensemble $LibreXYT_{o(i)}$ des paramètres permettant l'insertion de l'origine $o_i$ d'une demande $i$ sur une tournée $\Gamma_k$ et l'ensemble $LibreXYT_{d(i)}$ permettant l'insertion de la destination $d_i$ sur une tournée $\Gamma_{k'}$.

Pour tout $i \in D$, l'ensemble $LibreXYT_{o(i)}$ est donc constitué des paires $(k, z)$ pour lesquelles l'expression booléenne donnée en 5.4 est de valeur VRAI. Cette expression utilise un nœud actif $z \in \Gamma_k$, tel que $z \neq Queue(\Gamma_k)$, le successeur $z'$ de $z$ dans $\Gamma_k$, leurs fenêtres de temps courantes $[a(z),b(z)]$, $[a(z'),b(z')]$, et enfin la fenêtre de temps de $o_i$, $[\alpha, \beta]$ :

$$\begin{array}{c} (a(z) + DIST(z, o_i) \leq \beta) \wedge (\alpha + DIST(o_i, z') \leq b(z')) \\ \wedge (a(z) + DIST(z, o_i) + DIST(o_i, z') \leq b(z')) \wedge (ChT((\Gamma_k, z) + Q_i \leq CAP)) \end{array} \tag{5.4}$$

De la même manière, pour tout $i \in D$, l'ensemble $LibreXYT_{d(i)}$ est constitué des paires $(k, z)$ pour lesquelles la seconde expression booléenne 5.5 sera VRAI. Celle-ci utilise là encore un nœud actif $z \in \Gamma_k$, tel que $z \neq Queue(\Gamma_k)$, son successeur $z'$, les fenêtres de temps $[a(z), b(z)]$, $[a(z'), b(z')]$ et enfin la fenêtre de temps de $d_i$, $[\alpha, \beta]$ :

$$\begin{array}{c} (a(z) + DIST(z, d_i) \leq \beta) \wedge (\alpha + DIST(d_i, z') \leq b(z')) \\ \wedge (a(z) + DIST(z, d_i) + DIST(d_i, z') \leq b(z')) \wedge (ChT((\Gamma_k, z) + Q_i \leq CAP)) \end{array} \tag{5.5}$$

**Création du nœud relais**

Soit une demande $i \in D$ à insérer via $INSERTT$, dans 2 tournées cibles $\Gamma_{k_1}$ et $\Gamma_{k_2}$. On suppose que chaque tournée est datée par une fonction $t$, héritée des itérations précédentes. On suppose enfin que sont fournies $x_1 \in \Gamma_{k_1}$ et $x_2 \in \Gamma_{k_2}$ tels que $(x_1, k_1) \in LibreXY_{o_i}$ et $(x_2, k_2) \in LibreXY_{d_i}$.

On se propose donc de spécifier 3 valeurs $y_1$, $x_2$, $u$, de telle sorte que l'application de $INSERTT(i,k_1,k_2,x_1,y_1,x_2,y_2,u)$ devienne réalisable. Pour ce faire, nous posons, pour tout $y$ dans $\Gamma_{k_1}$ :

— Fermeture($y$, $k_2$) = [premier élément $x$ de $\Gamma_{k_2}$ tel que $t(x) \geq t(y)$].

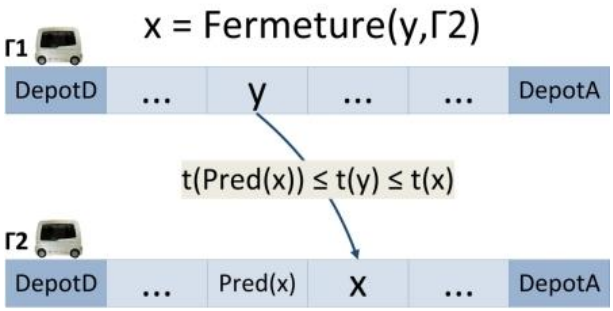


FIGURE 5.3 – exemple de Fermeture

La procédure Echange procède alors comme suit :

**Algorithme 18** Echange

**ENTRÉES:** $x_1$, $k_1$, $y_2$, $k_2$

**SORTIES:** les deux points d'ancrage et le nœud relais $y_1$, $x_2$, $u$

1: calculer $y_1$ dans $\Gamma_{k_1}$ tel que : { $x_2 \leftarrow Pred(Fermeture(y_1, k_2))$ ; $x_1 <<^{=}_{\Gamma_{k_1}} y_1$ ; $x_2 <<^{=}_{\Gamma_{k_2}} y_2$ ; DIST($y_1$, $x_2$) est minimale } ;
2: **Si** le calcul a échoué **Alors**
3: **Retourner** (Nil,Nil,Nil) ;
4: **Sinon**
5: **Retourner** $(y_1, x_2, MILIEU(y_1, x_2))$ ;
6: **Fin si**

La procédure Echange fournit alors le triplet $(y_1, x_2, u)$ souhaité.

**Remarque : la fonction MILIEU.** Elle va constituer de fait une partie de l'INPUT du DARPT, constituant l'interface entre le réseau réel auquel appartiennent tant les nœuds $o_i$, $d_i$, $i \in D$ que les nœuds relais de $u$, et le réseau virtuel fermé de $X$ et des nœuds $(u, i, \epsilon)$ actifs. Deux nœuds courants de $Z^* = X \cup U^*$ étant donnés, MILIEU produit un nouveau nœud $u \in U$ et ses 2 images $(u, i, 1)$, $(u, i, -1)$ dans $U^*$, et étend la matrice DIST pour tenir compte de ces nouveaux nœuds. Elle opère de façon heuristique, implémentant une version empirique de la notion de milieu, et son contenu différera selon que l'on travaille sur un réseau urbain, fait de nœuds et d'arcs, ou sur espace continu s'apparentant par exemple au plan euclidien.

**Faisabilité d'une application de INSERTT**

Un jeu de paramètres $(i, k_1, k_2, x_1, y_1, x_2, y_2, u)$ ayant été identifié pour INSERTT, nous devons tester la faisabilité de son application, et donc sa capacité à produire un système de tournées *admissible*.

Le fait que la *charge validité* soit maintenue pour $\Gamma_1$ et $\Gamma_2$ se teste exactement sur le même mode que dans le chapitre 3 sur le DARP Standard. Par contre, tester le maintien de la *temps validité* est plus délicat, car l'existence des nœuds d'échange ($u$,$i$,$\epsilon$) interdit de limiter l'analyse de *temps validité* aux seules tournées $\Gamma_1$ et $\Gamma_2$. De fait, nous devons tenir compte de ce que nous nommons ici les "contraintes liantes", à savoir, les contraintes :

— pour tout $(u, i, -1)$ actif, on a $t(u, i, -1) \leq t(u, i, 1)$ ;
— pour tout $i$ tel que $o_i$ et $d_i$ sont dans 2 tournées distinctes : $t(o_i) \leq t(d_i) \leq t(o_i) + \Delta_i$ ;

En termes de règles d'inférences, cela se traduit par le fait, qu'aux 5 règles d'inférences décrites en chapitre 3 s'ajoutent les règles R6 et R7 suivantes :

— **R6** :
$$\begin{array}{c} y = Jumeau(x); Statut(x) = ChargeOut; \\ FP.min(x) > FP.min(y) \\ \models \\ FP.min(y) \leftarrow FP.min(x) \end{array}$$

— **R7** :
$$\begin{array}{c} y = Jumeau(x); Statut(x) = ChargeOut; \\ FP.max(x) > FP.max(y) \\ \models \\ FP.max(x) \leftarrow FP.max(y) \end{array}$$

Les règles R3 et R4 du chapitre 3 doivent alors s'écrire :

— **R3** :
$$\begin{array}{c} y = Jumeau(x); Statut(x) = origine \text{ ou } DepotD; \\ FP.min(x) < FP.min(y) - \Delta(x) \\ \models \\ FP.min(x) \leftarrow FP.min(y) - \Delta(x) \end{array}$$

— **R4** :
$$\begin{array}{c} y = Jumeau(x); Statut(x) = origine \text{ ou } DepotD; \\ FP.max(y) > FP.max(x) + \Delta(x) \\ \models \\ FP.max(y) \leftarrow FP.max(x) + \Delta(x) \end{array}$$

La figure 5.4 illustre l'application de ces nouvelles règles.

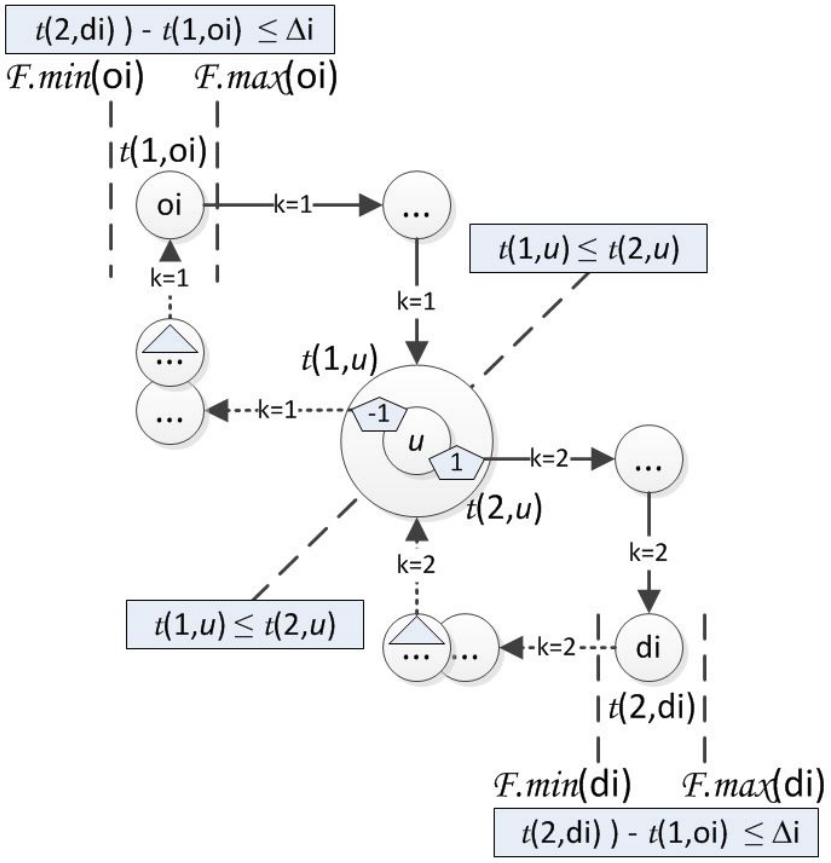


FIGURE 5.4 – Transbordement d'une demande avec respect des contraintes de temps

La procédure *PropageT* de la page suivante traduit sous forme algorithmique le processus de propagation induit. Elle peut-être vue comme une extension simple de la procédure de propagation du chapitre 3, adaptée à la prise en compte des règles R6 et R7.

Nous pouvons énoncer la proposition suivante (démonstration laissée au lecteur) :

**Proposition 1**

La collection de tournées $\Gamma$ représente un système de tournées *admissible* si et seulement si *PropageT* renvoie le booléen *Rejet* = FAUX. Les valeurs $t(x) = FP.min(x), x \in \Gamma$, forment alors un ensemble de dates $t$ *temps valides*.

**Algorithme 19** PropageT

**Entrées:** $\Gamma$, $L$, $FP(x)$ avec $x \in \Gamma$

**Sorties:** le booléen $Rejet$ et l'ensemble des fenêtres $FP$

$L \leftarrow \{x_1, y_1, x_2, y_2, (u, i_0, -1), (u, i_0, 1)\}$ ; $Rejet \leftarrow FAUX$ ;
**Tant que** $(L \neq NIL) \wedge (Rejet = FAUX)$ **Faire**
  $x \leftarrow Tete(L)$ ; $y \leftarrow Succ(x)$ ; $z \leftarrow Pred(x)$ ;
  $u \leftarrow Jumeau(x)$ ; $s \leftarrow statut(x)$ ; $L \leftarrow ListeSansTete(L)$ ;
  **Si** $(y \neq NIL) \wedge (FP.min(x) + DIST(x, y) > FP.min(y))$ **Alors**
    $FP.min(y) \leftarrow FP.min(x) + DIST(x, y)$ ; insérer $y$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(y \neq NIL) \wedge (FP.max(y) - DIST(x, y) < FP.max(x))$ **Alors**
    $FP.max(x) \leftarrow FP.max(y) - DIST(x, y)$ ; insérer $x$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(z \neq NIL) \wedge (FP.min(z) + DIST(z, x) > FP.min(x))$ **Alors**
    $FP.min(x) \leftarrow FP.min(z) + DIST(z, x)$ ; insérer $x$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(z \neq NIL) \wedge (FP.max(x) - DIST(z, x) < FP.max(z))$ **Alors**
    $FP.max(z) \leftarrow FP.max(x) - DIST(z, x)$ ; insérer $z$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(s = Origin) \wedge (FP.min(u) - \Delta(x) > FP.min(x))$ **Alors**
    $FP.min(x) \leftarrow FP.min(u) - \Delta(x)$ ; insérer $x$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(s = Origin) \wedge (FP.max(x) + \Delta(x) < FP.max(u))$ **Alors**
    $FP.max(u) \leftarrow FP.max(x) + \Delta(x)$ ; insérer $u$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(s = Destination) \wedge (FP.min(x) - \Delta(x) > FP.min(u))$ **Alors**
    $FP.min(u) \leftarrow FP.min(x) - \Delta(x)$ ; insérer $u$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(s = Destination) \wedge (FP.max(u) + \Delta(x) < FP.max(x))$ **Alors**
    $FP.max(x) \leftarrow FP.max(u) + \Delta(x)$ ; insérer $x$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(s = ChargeOut) \wedge (FP.max(u) < FP.max(x))$ **Alors**
    $FP.max(x) \leftarrow FP.max(u)$ ; insérer $x$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(s = ChargeOut) \wedge (FP.min(u) < FP.min(x))$ **Alors**
    $FP.min(u) \leftarrow FP.min(x)$ ; insérer $u$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(s = ChargeIn) \wedge (FP.max(u) > FP.max(x))$ **Alors**
    $FP.max(u) \leftarrow FP.max(x)$ ; insérer $u$ dans $L$ s'il n'y est pas ;
  **Fin si**
  **Si** $(s = ChargeIn) \wedge (FP.min(x) < FP.min(u))$ **Alors**
    $FP.min(x) \leftarrow FP.min(u)$ ; insérer $x$ dans $L$ s'il n'y est pas ;
  **Fin si**
  si $FP.min(x) > FP.max(x)$ alors $Rejet \leftarrow VRAI$ ;
**Fin tant que**
**Retourner** $(Rejet, FP)$ ;

**Evaluation d'une insertion INSERTT**

Les jeux de paramètres $(i, k_1, k_2, x_1, y_1, x_2, y_2, u)$ rendant possible l'insertion de la demande $i \in D$ dans le système de tournées $\Gamma$ via la procédure INSERTT ayant été identifiés, nous devons choisir celui qui donnera lieu à l'insertion effective via l'application d'un processus d'évaluation. La présence des nœuds relais et des processus d'échange fait, là encore, que ce processus d'évaluation concerne l'ensemble du système $\Gamma$, et non pas une tournée cible $\Gamma_k$. Nous renonçons dès lors à l'utilisation, dans le cadre de ce processus de choix de paramètres, d'un processus du type de *EvalTour2* du chapitre 3, et nous nous limitons à adapter le processus *EvalTour1* de ce même chapitre.

Cette adaptation débouche sur l'algorithme *EvalTourT* de la page suivante, qui opère à la façon d'un algorithme de Bellman, initialisant les dates $t(x), x = DepotD_k, k = 1..K$, sur leurs valeurs maximales $FP.max(x)$, puis propageant de proche en proche les contraintes :

— si $y = Succ(x, \Gamma_k)$ alors $t(y) \geq t(x) + DIST(x, y)$,

— $t(u, i, 1) \geq t(u, i, -1)$.

De cette façon, nous assignons aux autres valeurs $t(x), x \notin \{DepotD_k, k = 1..K\}$, les plus petites valeurs possible. Le processus *EvalTourT* ainsi spécifié, tend à minimiser la durée totale des tournées, c'est-à-dire le critère de performance $T_{Global}$. On peut en effet prouver :

**Proposition 2**

*EvalTourT* fournit un système de dates optimal dans le cas où B=C=0 et où seul $T_{Global}$ est impliqué dans la mesure de performance.

Dans le cas où B et C sont non nuls, la procédure *EvalTour2* du chapitre 3, trop lente pour être utilisée lors de la sélection du jeu de paramètre d'application de INSERTT, peut l'être pour le calcul final de la fonction de date $t$ associée au système de tournées $\Gamma$, et donc à la détermination de la performance finale effective de $\Gamma$.

**Remarque : situation d'interblocage.** Notre procédure de Bellman peut se bloquer quand le graphe défini sur les sommets actifs de $Z^*$ par les arcs $(x, Succ(\Gamma, x))$, $x \in \Gamma_k$, et $((u, i, -1),(u, i, 1))$ contient un circuit. Une telle situation, qui peut se produire sans correspondre à un cas d'impossibilité, est illustrée par la figure 5.5 de la page suivante.

Nous traitons ce type de situation, relativement rare, en considérant comme identiques les nœuds d'un tel circuit et en leur attribuant simultanément la même valeur $t$.

**Algorithme 20** EvalTourT

**Entrées:** une collection de tournées $\Gamma$.

**Sorties:** le résultat de l'évaluation $Eval(\Gamma, t)$ suivi de la collection de dates $t$.

**Pour tout** $x \in \Gamma$ **Faire**
  $[a(x), b(x)] \leftarrow FP(\Gamma)$ ; $Mark(x) \leftarrow 0$ ;
**Fin pour**
**Pour** $k$ de 1 à K **Faire**
  $t_{aux}(DepotD_k) \leftarrow b(DepotD_k)$ ;
  $Tab(k) \leftarrow DepotD_k$ ; $Mark(DepotA) \leftarrow 1$ ;
**Fin pour**
$k \leftarrow 1$ ;
**Tant que** $k \neq -1$ **Faire**
  $x \leftarrow Succ(Tab(k))$ ; $y \leftarrow Tab(k)$ ;
  **Si** $Statut(x) \neq ChargeIn$ **Alors**
    $Mark(x) \leftarrow 1$ ; $t_{aux}(x) \leftarrow Sup(a(x), t_{aux}(y)) + DIST(x, y))$ ;
    **Si** $Statut(x \neq DepotA)$ **Alors**
      $Tab(k) \leftarrow x$ ;
    **Sinon**
      $Index \leftarrow Index + 1$ ;
      $k \leftarrow$ {choisir un k tel que $Mark(DepotA) = 0$, $k \leftarrow -1$ sinon } ;
    **Fin si**
  **Sinon**
    **Si** $Mark(Jumeau(x)) = 1$ **Alors**
      $z \leftarrow Jumeau(x)$ ; $Mark(x) \leftarrow 1$ ;
      $t_{aux}(x) \leftarrow Sup(a(x), t_{aux} + DIST(y, x), t_{aux}(z))$ ; $Tab(k) \leftarrow x$ ;
    **Sinon**
      $k \leftarrow Dem(Jumeau(x))$ ;
    **Fin si**
  **Fin si**
**Fin tant que**
**Retourner** $(Eval(\Gamma, t), t_{aux})$ ;

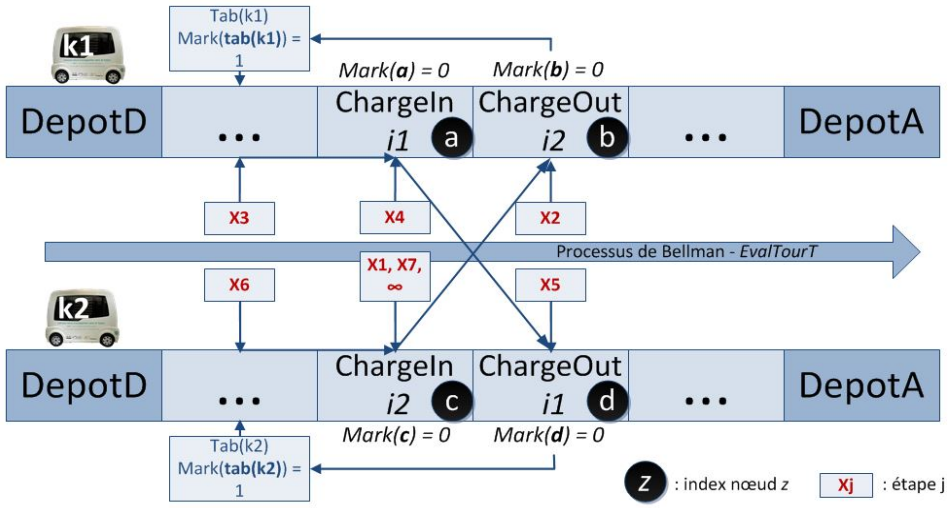


Figure 5.5 – Trace de 7 itérations $Xi$ du processus $EvalTourT$ en situation d'interblocage

### 5.2.3 Synthèse du processus de test et d'évaluation d'une insertion

Nous disposons d'une demande $i_0$ à insérer et d'un 7-*uplet* $(k_1,k_2,x_1,y_1,x_2,y_2,u)$. Dans un premier temps, nous testons la validité en termes de temps et de charge d'une insertion réalisée par l'opérateur *INSERTT* et, si aucun rejet ne se produit, nous évaluons cette insertion. La procédure $TestInsertionDARPT(i_0,k_1,k_2,x_1,y_1,x_2,y_2,u)$ qui réalise ce processus de test et d'évaluation, opère sur une copie de la collection de tournées $\Gamma$, et procède comme suit :

1. Elle teste la validité en termes de charge des deux tournées $\Gamma_{k_1}$ et $\Gamma_{k_2}$. Ceci n'implique pas la prise en compte d'autres tournées.
2. Les deux nœuds actifs jumeaux $(u, i_0, -1)$ et $(u, i_0 1)$ dans $Z^*$ sont créés et la matrice *DIST* est augmentée des distances séparant ces deux nœuds de leurs voisins sur les tournées $\Gamma_{k_1}$ et $\Gamma_{k_2}$.
3. L'opérateur $INSERTT(i_0,k_1,k_2,x_1,y_1,x_2,y_2,z)$ est appliqué à $\Gamma$. Les fenêtres de temps $[0;+\infty[$ sont affectées aux nœuds *Relais*. Puis la procédure *PropageT* est appliquée.
4. En cas de succès, c'est au tour de la procédure *EvalTourT* d'évaluer l'opération d'insertion.
5. La matrice *DIST* est restaurée tout comme $\Gamma$.

La procédure $TestInsertionDARPT(i_0,k_1,k_2,x_1,y_1,x_2,y_2,z)$ nous fournit un booléen de valeur VRAI si la validité des tournées de $k_1$ et $k_2$ est conservée, ainsi que le résultat de l'évaluation par *EvalTourT*.

### 5.2.4 Processus général de résolution

Nous sommes désormais capables d'écrire le processus d'insertions successives associé au problème de *Dial-a-Ride* avec transferts dans sa totalité. Ce processus opère en 2 phases :

— 1ère phase : la procédure *INSERTION* du chapitre relatif au *Dial-a-Ride* standard est appelée. Cette dernière nous renvoie un ensemble *Rejet1* d'index de demandes pour lesquelles l'insertion n'a pu se faire, ainsi qu'un système de tournées admissible $\Gamma$ et l'ensemble de dates $t$ associées aux demandes acceptées.

— 2ème phase : elle s'applique à partir de *Rejet*1 et du système ($\Gamma$, $t$) obtenu à l'issue de la première phase. Le processus induit *INSERTIONDARPT* consiste en une grande boucle *while* où chaque itération tente d'insérer une demande $i_0$ de *Rejet1* par l'opérateur *INSERTT* à l'aide des ensembles $LibreXYT_{o(i_0)}$ et $LibreXYT_{d(i_0)}$. Chacune de ces itérations se déroule comme suit :

1. $i_0$ est choisi aléatoirement parmi les demandes $i$ tel que $NbCarLibre_{o(i)} = 1$ ET $NbCarLibre_{d(i)} = 1$. Si de telles demandes n'existent pas, $i_0$ est choisie de façon aléatoire (en cas d'égalité) parmi les demandes $i$ minimisant la somme $NbCarLibre_{o(i)} + NbCarLibre_{d(i)}$ ; (E1)

2. un 7-*uplet* $(k_1,k_2,x_1,y_1,x_2,y_2,u)$ est mémorisé dans une liste *LCandidatsFaibles* s'il est vérifié que le couple $(k_1,x_1) \in LibreXYT_{o(i_0)}$, le couple $(k_2,x_2) \in LibreXYT_{d(i_0)}$ et que $(y_1,x_2,u) = Echange(x_1,k_1,y_2,k_2)$ ; (E2)
3. pour tout 7-*uplet* $(k_1,k_2,x_1,y_1,x_2,y_2,u)$ de la liste *LCandidatsFaibles*, la procédure $TestInsertionDARPT(i_0,k_1,k_2,x_1,y_1,x_2,y_2,u)$ est appelée. Les 7-*uplets* qui induisent un résultat positif sont insérés dans une liste notée *LCandidats*. Cette dernière est triée suivant l'ordre croissant des évaluations calculées par *EvalTourT* ; (E3)
4. Les demandes pour lesquelles *LCandidats* est vide sont rejetées et donc insérées dans un ensemble *Rejet* ; sinon l'insertion est possible et nous appliquons le processus suivant : (E4)
   (a) le 7-*uplet* effectif est tiré aléatoirement parmi les *N3* meilleurs éléments de *LCandidats*, $N3$ étant un paramètre de $INSERTIONDARPT$ ;
   (b) les nœuds actifs $(u,i_0,1)$ et $(u,i_0,-1)$ induits sont créés ;
   (c) l'insertion et la mise à jour des tournées deviennent effectives par application de la procédure $INSERTT(i_0,k_1,k_2,x_1,y_1,x_2,y_2,z)$,
   (d) les procédures *PropageT* et *EvalTourT* sont alors appelées afin de mettre à jour les fenêtres de temps et la fonction date $t$ ;
   (e) les listes $LibreXYT_{o(i_0)}$ et $LibreXYT_{d(i_0)}$ sont alors mises à jour pour toutes les demandes restantes de $J$.

**Algorithme 21** INSERTIONDARPT

**Entrées:** $\Gamma 1$ (partiellement construite par *INSERTION*), $t1$, $Rejet1$, $N3$
**Sorties:** $\Gamma$, $t$, $Eval(\Gamma,t)$, *Rejet*

$J \leftarrow Rejet1$ ; $Rejet \leftarrow NIL$ ; $\Gamma \leftarrow \Gamma 1$ ; $t \leftarrow t1$ ;
**Tant que** $J \neq NIL$ **Faire**
  sélectionner $i_0$ dans $J$ comme en (E1) et l'enlever de l'ensemble ;
  **Si** $(LibreXYT_{o(i_0)} = NIL)$ ou $(LibreXYT_{d(i_0)} = NIL)$ **Alors**
    $Rejet \leftarrow Rejet \cup \{i_0\}$ ;
  **Sinon**
    calculer *LCandidatsFaibles* selon (E2) et en déduire *LCandidats* selon (E3) ;
    **Si** $LCandidats = NIL$ **Alors**
      $Rejet \leftarrow Rejet \cup \{i_0\}$ ;
    **Sinon**
      sélectionner le 7-uplet $(k_1,k_2,x_1,y_1,x_2,y_2,u)$ de *LCandidats* selon (E4) ;
      créer deux nœuds actifs $(u,i_0,1)$ et $(u,i_0,-1)$ ;
      appliquer $INSERTT(i_0,k_1,k_2,x_1,y_1,x_2,y_2,u)$ ;
      mettre à jour $t$ et $FP$ par l'exécution de *PropageT* et de *EvalTourT* ;
    **Fin si**
    créer ou mettre à jour les ensembles $LibreXYT_{o(i)}$ et $LibreXYT_{d(i)}$ pour tout $i \in J$ ;
  **Fin si**
**Fin tant que**
**Retourner** $(\Gamma, t, Eval(\Gamma,t), Rejet)$ ;

**Processus Global DARPetDARPT**

Tel que décrit précédemment, l'enchaînement des processus INSERTION et INSERTIONDARPT a pour but de permettre la prise en compte de demandes que le mode de fonctionnement sans transfert tend à rejeter. Le point de vue qui le conditionne peut-être qualifié de défensif, dans la mesure où il cherche d'abord à éviter les situations d'échec. Dans l'hypothèse où l'application de INSERTION n'induit le rejet d'aucune demande, l'utilisation du mode avec transfert ne sera pas envisagée, même si celui-ci est susceptible de permettre une amélioration de la qualité des solutions. Afin de contourner cette difficulté, nous plongeons la séquence INSERTION → INSERTIONDARPT dans un processus plus global, qui agit sur les bornes $\Delta_i, i \in D$, et applique cette séquence à des jeux de bornes $\Delta_i^*, i \in D$, de plus en plus restrictifs, de façon à forcer les mécanismes de transfert. Pour chaque ensemble $\Delta_i^*, i \in D$, nous obtenons un système de tournées datées $(\Gamma, t)$, et, à l'issue du processus global, conservons la meilleure solution obtenue.

Le processus global DARPetDARPT induit se décrit dès lors en procédure 23 de la page suivante, la mise à jour des valeurs $\Delta_i^*, i \in D$, étant effectuée à l'aide d'un coefficient flexible $\lambda$, selon les termes de la procédure 22 ci-dessous.

**Algorithme 22** Mise à jour de $\Delta_i$

**Entrées:** $\lambda$ tel que $\lambda > 1$
**Sorties:** $\Delta_i^*, \forall i \in D$

1: **Pour tout** $i \in I$ **Faire**
2: **Si** $\Delta_i > \lambda.DIST(o_i, d_i)$ **Alors**
3: $\Delta_i^* \leftarrow \lambda.DIST(o_i, d_i)$ ;
4: **Sinon**
5: $\Delta_i^* \leftarrow \Delta_i$ ;
6: **Fin si**
7: **Fin pour**
8: **Retourner** les nouvelles bornes $\Delta_i^*, \forall i \in I$ ;

La figure 5.6 schématise le processus global :

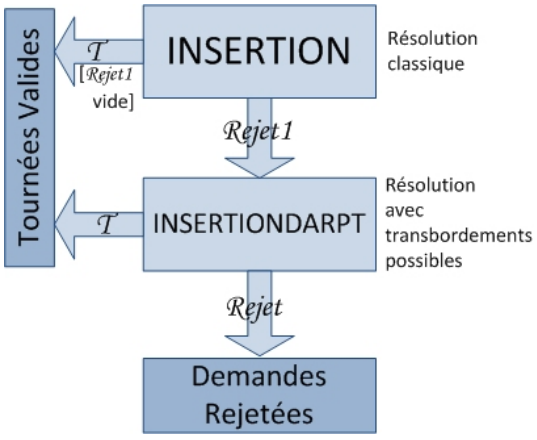


Figure 5.6 – Schéma général de résolution du DARP avec transferts

**Algorithme 23** DARPetDARPT

**Entrées:** un réseau de transit $G = (V, E)$, une flotte $VH$ de $K$ véhicules de capacité $CAP$, les pondérations du calcul de performance, l'ensemble des demandes $D = (D_i = (o_i, d_i, \Delta_i, F(o_i), F(d_i), Q_i, i \in I))$, les entiers $N1$, $N2$, $N3$ puis $P$ et enfin $\Lambda > 1$ ;

**Sorties:** la collection de tournées $\Gamma$ la plus performante associée aux dates de passage sur les nœuds $t$, la performance de la flotte $VH$ et l'ensemble des demandes rejetées $Rejet$ ;

1: $\lambda \leftarrow \Lambda$ ; $\Delta_{Aux_i} \leftarrow \Delta_i$ (sauver les bornes sur les temps de connexion) ;
2: $(\Gamma, t, Eval, Rejet)(P+1) \leftarrow (NIL, NIL, \infty, NIL)$ ;
3: **Pour** $p = 1..P$ **Faire**
4: mettre à jour des $\Delta_i$ pour toute demande $i$ de $D$ par l'algorithme 22 paramétré par $\lambda$ ;
5: $(\Gamma_1, t1, Eval1, Rejet1) \leftarrow INSERTION(N1, N2)$ ;
6: **Si** $Rejet1 = NIL$ **Alors**
7: $(\Gamma, t, Eval, Rejet)(p) \leftarrow (\Gamma_1, t1, Eval1, Rejet1)$ ;
8: **Sinon**
9: $(\Gamma, t, Eval, Rejet)(p) \leftarrow INSERTIONDARPT(\Gamma_1, t1, Rejet1, N3)$ ;
10: **Fin si**
11: **Si** $|(\Gamma, t, Eval, Rejet)(p).Rejet| < |(\Gamma, t, Eval, Rejet)(P+1).Rejet|$ **Alors**
12: $(\Gamma, t, Eval, Rejet)(P+1) \leftarrow (\Gamma, t, Eval, Rejet)(p)$ ;
13: **Sinon**
14: si $Rejet$ est équivalent, on sauve (ou garde) la meilleure $Eval$ ;
15: **Fin si**
16: $\lambda \leftarrow \lambda - (1/P).(\Lambda - 1)$ ;
17: **Fin pour**
18: $\Delta_i \leftarrow \Delta_{Aux_i}$ pour toutes les demandes $i$ de $D$ ;
19: **Retourner** $(\Gamma, t, Eval, Rejet)(P+1)$ ;

#### Quelques remarques d'ordre général

**Grand nombre de points relais.** Dans le cas où une solution viable met en jeu une grande quantité de nœuds relais, c'est-à-dire qu'elle utilise largement les transferts de chargement, il peut apparaître des situations incongrues pour les tournées a fortiori moins performantes. En effet, au sein du processus d'insertion d'une demande $i$, la création d'un nœud relais et la mise en place définitive des deux nœuds qui lui sont associés dans les deux tournées $k_1$ et $k_2$ ont pour but de minimiser les distances parcourues pour qu'elles se rejoignent. Les étapes suivantes vont considérer un tel nœud relais comme figé et aucun processus ne va chercher à modifier son emplacement malgré l'insertion de nouvelles demandes. Il serait de fait possible d'envisager un traitement complémentaire par recherche locale qui viserait à ré-optimiser la position des nœuds relais créés dans l'espace et dans les tournées, une fois celles-ci construites.

**Calibration des durées maximales de connexion.** A chaque itération $p$, $p = 1..P$, les durées maximales de connexion sont mises à jour par la formule d'affectation 5.6. Il est souhaitable que le programme soit paramétré par un $\Lambda > 1$ correctement sélectionné. Si ce dernier est trop petit, les dernières itérations seront inutiles en considération de plusieurs types de performance recherchée puisque les $\Delta_i, i \in D$

résultants seront beaucoup trop petits - jusqu'à être égaux à la distance les séparant si $\Lambda \simeq 1$ - impliquant ainsi une utilisation des véhicules sans partage. Inversement, si $\Lambda$ est trop grand, le nouveau $\Delta_i$ devient supérieur à sa valeur initiale et le programme garde alors cette dernière.

$$\lambda \leftarrow \lambda - (\frac{(\Lambda - 1)}{P}) \tag{5.6}$$

**Temps de service et transferts.** Nous avons vu, en chapitre 3, que des temps de service fixes (indépendants des chargements à effectuer et de l'état des véhicules) avaient vocation à être intégrés à la matrice des distances DIST et à ne pas figurer explicitement dans les modèles. Ce n'est pas forcément le cas si on considère des temps de service associés aux opérations de transfert. En effet, ceux-ci, même fixes, vont être corrélés au nombre d'opérations de transfert, et on peut considérer que, celles-ci induisant, tant pour l'opérateur que pour l'usager, des risques de mauvaise synchronisation, il est souhaitable de les pénaliser. Des temps de transfert pourraient remplir ce rôle de pénalités. Nous n'avons toutefois pas adopté ici ce point de vue.

## 5.3 Résultats expérimentaux

### 5.3.1 Objectifs

Ce chapitre se clôt sur des expérimentations mettant en relief le comportement de notre processus de résolution pour le problème DARP avec transferts autorisés. Cette section procède pour partie sur la même base que le chapitre précédent, à savoir la comparaison des résultats obtenus sur des mêmes instances résolues par notre heuristique résolvant le DARP puis celle relative au DARPT. Elle vise aussi à étudier les variations de la qualité des tournées induites par la procédure *DARPetDARPT*. Plus précisément, nous étudierons la sensibilité de la durée des tournées pour chaque variation de $\Delta_i, i \in D$.

La littérature du *Dial-a-Ride Problem* est pauvre en ce qui concerne les instances proposées, d'autant plus lorsque l'on intègre la caractéristique des transferts autorisés. Il est cependant possible de résoudre des instances classiques en intégrant des transbordements, mais l'intérêt de cette approche est limité par le fait que la plupart de ces instances sont conçues pour permettre, sans transfert, l'acceptation de toutes les demandes (pas de rejet).

Nous débuterons tout de même par une courte étude sur la première instance de [Cordeau and Laporte (2003)], et ceci afin d'étudier l'évolution des performances des solutions données par la résolution du DARP puis du DARPT, toutes les demandes pouvant à priori être insérées.

Nous étudierons donc ensuite les résultats obtenus par notre processus de résolution du DARPT sur des instances réalisées selon un schéma particulier : celui du *clustering* où des sous-flottes se partagent la satisfaction de demandes dans des régions qui leur sont propres (ou plutôt des régions où se trouvent leur dépôt respectif).

### 5.3.2 Environnement et performances

**Mesures de performance.** Nous reprenons la plupart des notations et mesures de performance des chapitres précédents. $T_{Insert_T}$ désigne le taux d'insertion global de l'algorithme $DARPetDARPT$, il est calculé avec l'ensemble $Rejet_T$ des demandes rejetées par ce même processus :

$$T_{Insert_T} = 100.\frac{|D| - |Rejet_T|}{|D|} \tag{5.7}$$

Nous relevons enfin le gap entre les taux d'insertion obtenus avec ou sans transfert :

$$Gap = 100.(T_{Insert_T} - T_{Insert})/T_{Insert} \tag{5.8}$$

**Méthodes programmées.** Les deux séries de tests exécutent l'heuristique *DARPetDARPT* (algorithme 23 développé tel quel). L'évaluation de la performance des tournées pour la partie DARPT est réalisée par la méthode *EvalTourT* (algorithme

20). Rappelons que $EvalTourT$ ne cherche qu'à minimiser la durée globale ($T_{Global}$) des tournées et traite donc le problème comme monocritère. Pour la résolution du DARP, avec l'algorithme INSERTION, nous n'avons utilisé qu'$EvalTour1$ qui minimise ce même critère.

**Environnement d'exécution.** Ces expérimentations ont fait l'objet d'un même environnement de développement (en C++) et d'un même environnement d'exécution (Linux à adressage 64 bits) que les chapitres précédents.

### 5.3.3 Les instances

**Série 1 : Instance standard de [Cordeau and Laporte (2003)]**

La première expérience sur le DARPT reprend une instance de [Cordeau and Laporte (2003)], avec comme objectif l'optimisation de la durée totale des tournées ($T_{Global}$). Nous observons simultanément l'évolution de $T_{Ride}$ en fonction des variations sur les bornes $\Delta_i$, telles que implémentées dans $DARPetDARPT$.

**Série 2 : Instances ad hoc *clusterisées***

La deuxième série de tests concerne des instances *clusterisées*, c'est-à-dire générées de telle sorte que les véhicules se partagent l'espace de déplacement, et échangent aux frontières des zones qui leur reviennent. Plus spécifiquement, on part d'un carré $\Re$ du plan euclidien, de longueur 20, partagé en 4 zones $EP_i, i = 1..4$, correspondant à 4 sous-flottes $VH_i, i = 1..4$, conformément à la figure suivante :


FIGURE 5.7 – Espaces prioritaires et dépôts de 4 sous-flottes

La distance DIST est donc la distance euclidienne. Contrairement à ce qui prévaut dans le modèle de base, nous considérons ici non pas un, mais 4 dépôts, respectivement situés sur les 4 extrémités du carré $\Re$. Chacun des K véhicules est rattaché à un des 4 dépôts. Les algorithmes décrits dans le présent chapitre sont donc adaptés, de façon triviale, à cette nouvelle hypothèse. Le processus de génération d'instances fonctionne alors comme suit :

— les nombres de véhicules $K_j, j = 1..4$, respectivement associés à chaque dépôt, ainsi que le nombre de demandes $|D|$, sont générés ;
— un premier paquet de demandes est généré, composé de $\alpha.|D|$ demandes dites "locales", c'est-à-dire connectant 2 points $o_i$, $d_i$, situés dans une même zone

$EP_i, i = 1..4$. Les volumes de demandes sont sensiblement identiques d'une zone à l'autre pour un total de 30% de l'ensemble des demandes de $D$ ;
— un deuxième paquet de $(1-\alpha)|D|$ demandes est généré, formé de demandes dites "transversales", c'est-à-dire connectant une origine et une destination situées dans 2 zones $EP_i, i = 1..4$, différentes. Elles représentent 70% des demandes de $D$ ;
— tout point du carré $\Re$ peut être vu comme un nœud relais.

Les données générées ont les propriétés suivantes :
— les charges $Q_i, i = 1..|D|$, sont unitaires et la capacité $CAP$ est égale à 6 ;
— les $o_i$, $d_i$, $i = 1..|D|$, sont générés selon une loi uniforme, conditionnés, dans le cas des demandes globales, par le fait qu'ils doivent se situer dans 2 zones $EP_i, i = 1..4$, différentes ;
— les valeurs $\Delta_i$ forment une contrainte large ;
— pour chaque demande $i$, $i = 1..|D|$, l'un des nœuds $o_i$ ou $d_i$ est doté d'une fenêtre de temps d'amplitude $EF$, où $EF$ est un paramètre du processus, variant entre 5 et 30 (les unités de temps coïncident ici avec les unités de distance). Dans tous les cas, on fait en sorte que, pour une demande donnée $i$, on ait : $F.min(d_i) \geq F.min(o_i) + DIST(o_i, d_i)$ et $F.max(d_i) \geq F.max(o_i) + DIST(o_i, d_i)$.

### 5.3.4 Résultats et analyse

**Série 1**

Le point le plus intéressant ressortant de cette expérience, est que le processus $DARPetDARPT$ permet implicitement une minimisation des temps de transit ($T_{Ride}$) des usagers sans que la performance $T_{Global}$ soit altérée. Cela découle des restrictions progressives imposées par le processus sur les bornes $\Delta_i, i = 1..|D|$. Nous pouvons remarquer sur le diagramme 5.8 que les temps de connexion sont pratiquement divisés par deux si l'on compare la première itération à la dernière (ces valeurs ont été relevées exclusivement sur des tournées valides où toutes les demandes de l'instance ont été insérées). La somme des durées des tournées ($T_{Global}$) demeure dans le même temps relativement stable.

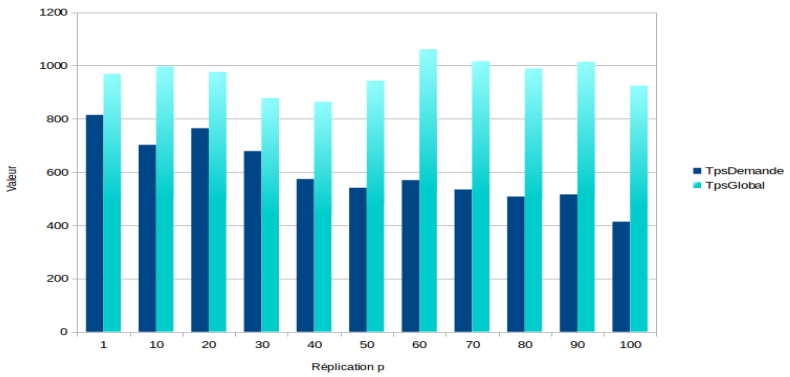


FIGURE 5.8 – Minimisation indirecte des temps de connexion - Instance pr01

**Série 2**

Nous reportons les résultats en tableau 5.1. Ils correspondent à 18 groupes de 10 instances où, pour chacune, les meilleurs taux (meilleure réplication) ont été relevés. On remarque que l'algorithme *DARPetDARPT*, autorisant les transferts, insère bien plus de demandes qu'*INSERTION*.

| $\|D\|$ | $K$ | $EF$ | $T_{Insert}$ | $T_{Insert_T}$ | $Gap$ |
|---|---|---|---|---|---|
| 32 | 4 | 5 | 38,8 | 52,7 | 35,8 |
| 32 | 4 | 15 | 54,6 | 62,8 | 15,0 |
| 32 | 4 | 30 | 61,8 | 68,3 | 10,5 |
| 32 | 5 | 5 | 49,7 | 70,7 | 42,1 |
| 32 | 5 | 15 | 70,3 | 86,0 | 22,4 |
| 32 | 5 | 30 | 77,0 | 89,1 | 15,7 |
| 64 | 4 | 5 | 21,2 | 28,5 | 34,7 |
| 64 | 4 | 15 | 29,4 | 34,3 | 16,9 |
| 64 | 4 | 30 | 35,3 | 37,1 | 5.0 |
| 64 | 5 | 5 | 27,6 | 38,7 | 40,1 |
| 64 | 5 | 15 | 39,1 | 46,1 | 18,1 |
| 64 | 5 | 30 | 45,2 | 48,2 | 6,7 |
| 96 | 4 | 5 | 15,7 | 20,9 | 33,0 |
| 96 | 4 | 15 | 22,4 | 24,2 | 7,7 |
| 96 | 4 | 30 | 25,9 | 27,0 | 4,2 |
| 96 | 5 | 5 | 20,5 | 28,2 | 37,9 |
| 96 | 5 | 15 | 28,6 | 32,6 | 13,9 |
| 96 | 5 | 30 | 33,3 | 35,4 | 6,3 |

TABLE 5.1 – Taux d'insertions sur 18 groupes d'instances

**Commentaires.** Nous pouvons remarquer que *Gap* connaît une baisse lorsque $|D|$ augmente. En effet, nous avons, pour les trois valeurs croissantes de $|D|$, respectivement un gap moyen de 23,58%, 20,25% et 17,54%. Ceci semble dû au fait que lorsque le nombre de demandes croît (notamment les demandes non-transversales), la flotte perd en taux d'insertion mais, que ce soit pour l'une ou l'autre des résolutions, remplit plus facilement ses véhicules. Dans les deux résolutions, les demandes transversales sont, au fil d'une réplication, insérées en fin de processus puisque la minimisation des durées des tournées implique de prendre en premier lieu les demandes autour du dépôt (créant ainsi un phénomène automatique de *clustering*). Lorsque le processus commence à intégrer les demandes transversales, les véhicules sont déjà bien remplis. C'est pourquoi plus il y a de demandes locales moins les véhicules auront la capacité de gérer les demandes transversales.

La variation sur $K$ semble montrer que l'augmentation du nombre de véhicules profite aux transferts (gap moyen de 18,09% pour 4 véhicules et de 22,58% pour 5 véhicules). De manière inverse, ceci s'explique comme la diminution de *Gap* lorsque les demandes augmentent, ici l'espace d'accueil croissant, les possibilités de remplissage sont plus importantes. Cette différence peut également provenir d'un autre élément : le nombre de combinaisons entre les véhicules créant un point de transfert

est plus important.

La figure 5.9 est un cliché de ces résultats avec un ensemble de tournées avec transferts provenant d'une instance de petite taille.

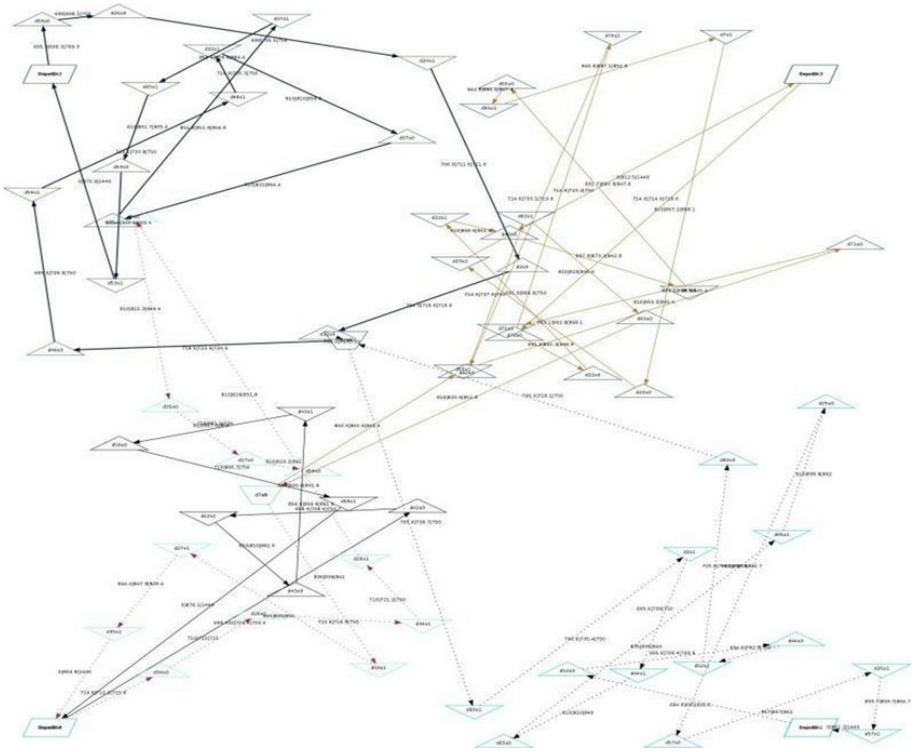

FIGURE 5.9 – Rendu du processus de *clustering*

## Perspectives et conclusion sur le DARPT

Nous avons mis au point une heuristique à base d'insertions successives pour résoudre les problèmes de *Dial-a-Ride* avec la possibilité de réaliser un transfert. Notre algorithme est suffisamment générique pour utiliser le même principe pour un nombre plus élévé de transbordements possibles dans la satisfaction d'une demande donnée (mais la longueur des algorithmes en aurait été grandement affectée). Au moment de tester une insertion, la propagation de contraintes du chapitre 3 intègre deux nouvelles règles d'inférences permettant de couvrir l'ensemble des tournées. Ainsi, les contraintes concernant les durées maximales d'acheminement mais aussi celles venues des fenêtres de temps sont toujours respectées si l'algorithme valide une insertion. La difficulté suivante a été de déterminer le moment où les transferts pouvaient contribuer à la résolution du problème DARP. Nous avons mis au point un second algorithme contractant les durées maximum d'acheminement au fil des réplications pendant lesquelles les résolutions du DARP et du DARPT s'enchaînaient (s'il y a rejet des demandes par le premier). Les prochaines évolutions de l'algorithme pourront améliorer la création de ces nœuds par un traitement final, c'est-à-dire une fois que l'on connaîtra les chemins pris par les flottes (comme par exemple, une recherche locale dont l'opérateur appliquera des permutations avec les nœuds relais).

A nouveau, mais cette fois-ci dans le cadre du DARPT, il sera nécessaire de considérer l'arrivée des demandes qui doivent être traitées par des véhicules déjà en train de satisfaire des demandes calculées d'autres optimisations précédentes et de les intégrer dynamiquement. Là encore, notre algorithme permet que cela se fasse le plus facilement possible.

A ce moment de notre étude, il est difficile de ne pas envisager l'hypothèse d'un rapprochement entre les modules *division du chargement* et *transbordement* dans une nouvelle variante du DARP. La manière la plus simple consisterait à modifier *INSERTIONDARPT* dans sa première étape en remplaçant *INSERTION* par *INSERTIONDARPSL* : de cette manière, les transferts viendraient en auxiliaires de la division du chargement et non plus de la résolution classique. La figure 5.10 actualise le processus général ainsi mis à jour. Il serait également possible de traiter l'ensemble *Rejet*1 afin de distribuer les tournées ainsi que les demandes non insérables ainsi en fonction de leurs caractéristiques (cf. figure 5.11).

Enfin, le *framework* mis au point pourrait encore faire l'objet de nombreuses expérimentations. Nous pourrions par exemple reprendre le problème de *clustering* et nous en servir afin de situer les différents dépôts aux endroits stratégiques qui impliquent la meilleure performance et ceci selon les types de distribution de demandes. Autrement dit, nous pourrions distribuer les demandes selon un scénario donné puis modifier au fil des réplications la place des différents dépôts de chaque sous-flotte pour finalement en déterminer leur meilleur emplacement. Le même schéma pourrait également se faire sur la distribution des populations de véhicules dans les sous-flottes et, aussi, sur la distinction des dépôts de départ et d'arrivée.

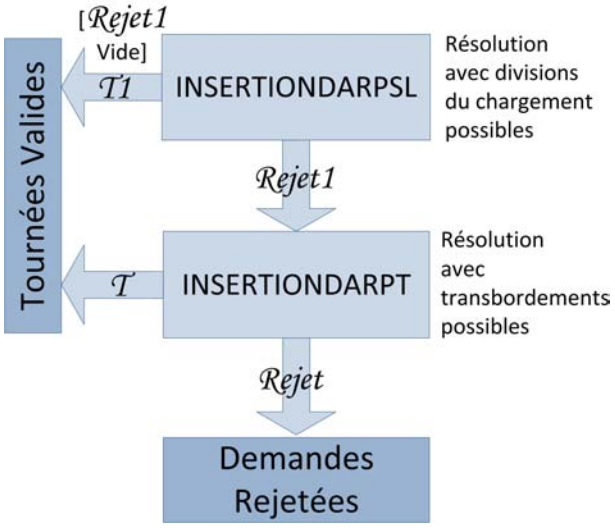


FIGURE 5.10 – Intégration de la division de charge au schéma général de résolution

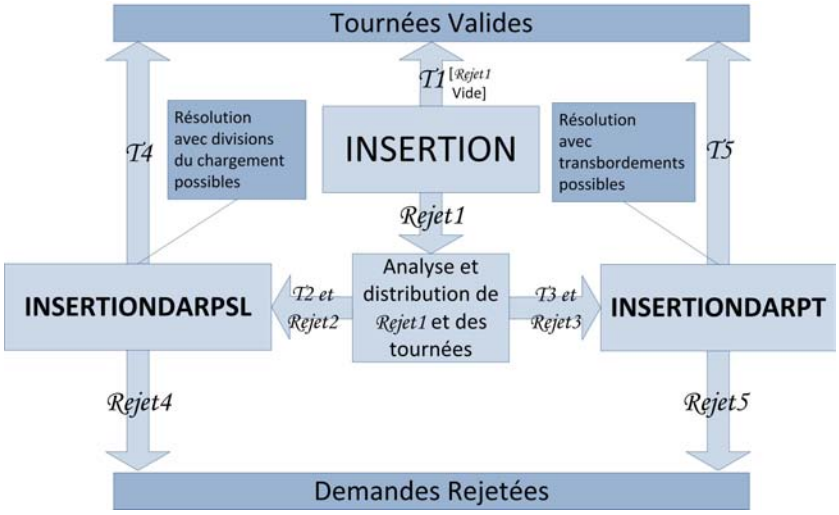


FIGURE 5.11 – Schéma général avec les 3 heuristiques de résolution

# Chapitre 6

# Introduction à la robustesse dans le DARP dynamique - DARPRob

## Introduction

Dans le problème de *Dial-a-Ride* statique, l'ensemble des données est connu à l'avance alors qu'un service de transport à la demande nécessite l'intégration des requêtes des usagers au fur et à mesure et cela de manière réactive. Ce contexte induit le problème dit du DARP dynamique (dont l'état de l'art a été écrit dans le chapitre 2). Les algorithmes étudiés jusque là permettent un passage relativement aisé à cet autre problème. Mais, les traitements du DARP par insertions successives des demandes, tels qu'ils ont pu être réalisés jusqu'à présent, souffrent d'un défaut : quand une demande est insérée dans un ensemble de tournées courantes, il est peu tenu compte de l'impact de cette insertion sur la difficulté qu'il pourrait y avoir à traiter les demandes restantes. De fait, le seul moment où une telle prise en compte a lieu se situe dans le choix de la demande à insérer, avec priorité faite à celles pour lesquelles l'insertion tend à devenir difficile. Pour le reste, le choix des paramètres d'insertion (la tournée puis les positions de l'origine et de la destination au sein de celle-ci) ne dépendent que de celles qui sont déjà réalisées.

Au fil des insertions, la capacité d'accueil des tournées par rapport aux contraintes de temps tend à se rétrécir. Ceci se manifeste au travers de la propagation de contraintes appliquée aux différents points de passage déjà attribués aux véhicules par les 5 règles d'inférence décrites dans le chapitre 3. En contexte dynamique, ce point est d'autant plus critique que des rendez-vous avec les usagers doivent être fixés (cf. figure 6.1 page suivante).

Prenons par exemple deux nœuds consécutifs d'une même tournée qui ont été insérés lors d'un premier processus de résolution. Supposons que ces deux insertions ont été réalisées de façon à minimiser la distance totale parcourue. Si les contraintes relatives aux demandes de ces nœuds le permettent, le temps qui séparera ces deux points de passage sera exactement celui qui est nécessaire pour parcourir la distance qui les sépare. Plus tard, lors d'un second processus de résolution, l'insertion d'une tierce demande entre ces deux rendez-vous sera difficile à appliquer, voire impossible.

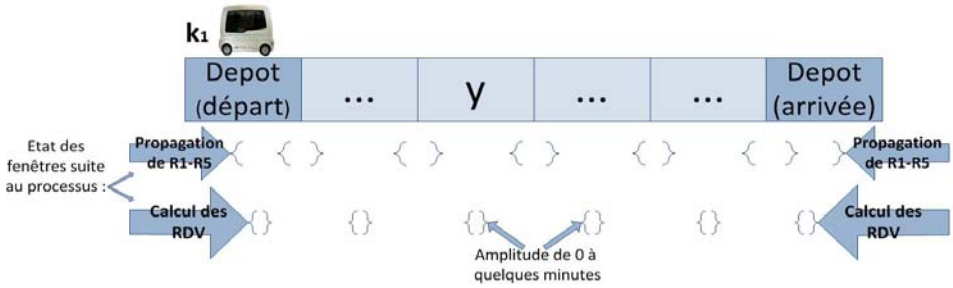


FIGURE 6.1 – Contraction des fenêtres de temps

Plus précisément, ne pourra être inséré aucun nœud qui impliquerait des temps de service, ou qui ne se situerait pas sur le segment rejoignant les deux premiers, là où les rendez-vous ont déjà été pris.

Nous reprenons dans ce chapitre les caractéristiques des précédents, nous intégrons une nouvelle fois une flotte de véhicules à capacité finie et commune prenant en charge des demandes définies par une charge, une géographie, une borne sur la durée de transit et des fenêtres de temps. D'abord pour le DARP statique puis pour sa version dynamique, notre but est de montrer comment, au moment de réaliser une insertion, il est possible de prendre en compte les demandes à venir afin de leur laisser suffisamment de place, dans un souci de rendre le système plus robuste (pour le cas dynamique).

Nous introduisons d'abord une mesure d'*Insérabilité* - ang. *Insertability* - pour chaque demande non encore prise en charge. Cette mesure met en jeu l'amplitude des fenêtres de temps. Nous l'utilisons de 2 façons : en contexte statique afin de faciliter l'acceptation des demandes et, en contexte dynamique, afin de préserver les possibilités d'insertion des demandes futures.

Dans un premier temps, pour le problème DARP statique, nous voyons comment, au moment d'insérer une demande, anticiper les insertions grâce à cette mesure. L'*Insérabilité* nous permet notamment de déterminer si une demande doit être rejetée, dès lors que son acceptation entraînerait une importante chute de l'*Insérabilité* des demandes non encore insérées. Une première série d'expérimentations permet de tester et d'étudier le comportement de ces techniques.

Dans un second temps, nous adaptons ces procédés au cas dynamique. Ici, nous anticipons sur les demandes futures qui seront probablement à insérer lors de prochaines exécutions du processus de résolution. Nous représentons ces demandes futures grâce à la notion de demandes virtuelles. Nous tendons donc à pratiquer les insertions courantes de façon à préserver l'*Insérabilité* de ces demandes virtuelles ce qui revient à introduire un critère de Robustesse. Pour permettre une seconde série d'expérimentations adaptées au contexte dynamique, nous mettons en place un modèle de simulation.

## 6.1 Anticipation dans le DARP statique - Notion d'*Insérabilité*

On considère à nouveau un ensemble de demandes $D$ où chaque demande $i$, est définie par une origine $o_i$, une destination $d_i$, des fenêtres de temps $F(o_i)$ et $F(d_i)$, une limite sur la durée d'acheminement $\Delta_i$ et une charge $Q_i$. A cet ensemble est associé un ensemble de nœuds $X = \cup_{i \in D}\{o_i, d_i\} \cup [\cup_{k=1..K}\{DepotD_k, DepotA_k\}]$ ainsi qu'une matrice des distances $DIST$ sur $X.X$. Les éléments de $X$ sont considérés comme étant tous distincts, même s'ils recouvrent une même réalité géographique. On veut, à l'aide d'un nombre $K$ de véhicules identiques, traiter ces demandes, en respectant les contraintes de fenêtres, de durées d'acheminement et de capacité induites, sachant que la durée des tournées est bornée. A tout instant, durant le processus d'insertion, on dispose de :

- l'ensemble $D - D1$ des demandes déjà insérées, $D1$ étant l'ensemble des demandes à prendre en charge ;
- un ensemble $\Gamma = \{\Gamma_k,\ k = 1..K\}$ de tournées, $K$ étant le nombre de véhicules utilisés. Chaque tournée $k$ démarre en $DepotD_k$ et se finit en $DepotA_k$, et traite les demandes de $D - D1$ en respectant l'ensemble des contraintes qui leurs sont imposées ;
- un état courant des fenêtres $F(x)$, $x \in \cup_k \Gamma_k$ de temps encore autorisées pour les différents nœuds de passage des tournées courantes. Ces fenêtres sont évolutives et tendent à se contracter à chaque insertion ;
- une liste $NbCarLibre(i)$, pour chaque demande $i$ restante à insérer, des véhicules dans lesquels cette insertion peut encore avoir lieu, et, pour chacun de ces véhicules, de l'ensemble *LibreXY(i)* des 4-uplets $(k, x, y, v)$ où l'on retrouve les couples $(x, y)$ d'insertion possible : $x$ et $y$ sont deux nœuds de $\Gamma_k$, tels qu'il est possible d'insérer $o_i$ entre $x$ et son successeur dans $\Gamma_k$ (noté $Succ(\Gamma_k, x)$) et $d_i$ entre $y$ et $Succ(\Gamma_k, y)$, ainsi qu'une estimation $v$ de la valeur de l'insertion envisagée. Cette estimation de valeur ne porte que sur les critères standards de qualité des tournées (temps global, temps d'acheminement, temps d'attente).

### 6.1.1 Mesure d'*Insérabilité* et optimisation de l'*Insérabilité*

Le but de cette section est d'introduire un outil évaluant l'impact qu'a l'insertion d'une demande sur les possibilités d'insertion des demandes restantes. Pour cela, nous introduisons la fonction $INSER(i, \Gamma)$ mesurant la capacité d'insertion d'une demande $i$ dans une collection de tournées $\Gamma$, c'est à dire son *Insérabilité* :

- $INSER(i, \Gamma) = \sum_{k \in K} INSER1(i, \Gamma_k)$ ;
- $INSER1(i, \Gamma_k) = Max_{x,y} INSER2(i, \Gamma_k, x, y)$, pour toute tournée $\Gamma_k$, $x$ et $y$ désignant ici les possibles nœuds d'ancrage pour l'insertion de l'origine et de la destination de la demande $i$ dans $\Gamma_k$ ; $INSER1(i, \Gamma_k)$ est donc associée aux paramètres $x, y$ qui offrent la plus grande capacité d'insertion *INSER2* de $i$ dans la tournée $\Gamma_k$ ;
- $INSER2(i, \Gamma_k, x, y) = (FP.max(o_i) - FP.min(o_i)).(FP.max(d_i) - FP.min(d_i))$, $FP$ désignant les fenêtres de temps, associées à $o_i$ et $d_i$, et contractées suite à

la propagation de contraintes résultant de l'insertion de la demande $i$ dans la tournée $\Gamma_k$.

Ces fonctions sont définies de manière empirique. *INSER2* est calculée sur le produit de l'amplitude des deux fenêtres, mais d'autres opérateurs que le produit auraient pu être choisis.

#### Optimisation de l'*Insérabilité* d'un ensemble de demandes

$INSER(i, \Gamma)$ évalue la facilité qu'il y a, à l'instant considéré, à insérer $i$ dans $\Gamma$. Nous allons utiliser les quantités $INSER(i, \Gamma), i \in D1$, afin de calculer, pour une demande cible $i_0$ à insérer, la tournée cible et les paramètres d'insertion associés.

Soit donc $\Gamma = (\Gamma_k,\ k = 1..K)$ l'ensemble des tournées courantes, $i_0 \in D1$, la demande cible, identifiée selon les mécanismes en vigueur dans le cas standard, et $D1^* = D1 - \{i_0\}$, l'ensemble des demandes qu'il restera à insérer une fois $i_0$ traitée. Insérer $i_0$ de façon à optimiser l'*Insérabilité* des demandes restantes dans $D1^*$ revient à :

{Sélectionner la tournée $k$ et les deux points d'ancrage $x$ et $y$, de telle sorte que la quantité résultante $Min_{i \in D1^*} INSER(i, Inserted(\Gamma, i_0, k, x, y))$ soit la plus grande possible, $Inserted(\Gamma, i_0, k, x, y)$ désignant le nouvel ensemble de tournées $\Gamma$ obtenu après insertion de $i_0$ dans $\Gamma_k$ selon les nœuds d'ancrage $x$ et $y$.}

Idéalement, choisir $k$, $x$, $y$ de façon à optimiser l'*Insérabilité* de $D1^*$ aura pour effet de faciliter la recherche d'une solution réalisable du DARP Standard. Dans la pratique, il faudra de fait combiner ce critère d'*Insérabilité* avec les autres critères de qualité de l'insertion : durée globale des tournées, durée d'acheminement des demandes, durée des attentes...). Un point sensible, non abordé jusque là, tient à la prise en compte de l'éventualité du rejet de certaines demandes : si l'on admet qu'il n'est pas possible de traiter toutes les demandes, alors la stratégie du choix de la demande cible est susceptible d'être remise en cause. Il peut en effet s'avérer plus efficace de faire l'impasse sur une demande qui semble très difficile à traiter que de la placer en priorité, comme cela a été le cas jusqu'à présent, au risque de rendre plus difficile l'insertion des autres demandes. On verra que, dans cette perspective, la mesure $INSER(i, \Gamma)$ peut également être utilisée pour caractériser le rejet d'une demande.

### 6.1.2 Adaptation des procédures du DARP standard à la prise en compte de l'*Insérabilité*

Le chapitre 3 décrit un algorithme par insertion et propagation de contraintes résolvant le DARP standard. Nous l'adaptons à la prise en compte de l'*Insérabilité*. L'heuristique comporte deux niveaux de décision : la sélection de la demande à insérer et celle des paramètres de son insertion (tournée et emplacement).

#### Sélection de la demande à insérer

Le choix de $i_0$, la demande à insérer, se fait, selon l'algorithme du chapitre 3, sur la base de la cardinalité de l'ensemble *NbCarLibre* associé. Ce choix est affiné par la prise en compte des valeurs $INSER(i, \Gamma)$ : une demande $i$ devient prioritaire pour une insertion si la quantité $INSER(i, \Gamma)$ est faible.

#### Sélection des paramètres d'insertion

On part donc de la situation courante décrite précédemment en considérant qu'une demande cible $i_0$ a été identifiée. On veut alors calculer les paramètres $k$, $x$, $y$ pour l'insertion de $i_0$. L'algorithme 24 montre comment il est possible de calculer ces paramètres via l'optimisation de l'*Insérabilité*, selon les termes du paragraphe précédent.

**Algorithme 24** Sélection des paramètres d'insertion par la mesure de l'*Insérabilité*

**Entrées:** la collection courante $\Gamma$ de tournées, la demande sélectionnée $i_0$, les ensembles *libreXY* de chaque demande restante (dont $i_0$) ;

**Sorties:** le triplet $(x, y, k)$ correspondant aux paramètres d'insertion de $i_0$ optimisant les possibilités insertion des demandes restantes de *D1*-$\{i_0\}$ ;

1: $maxMinINSERXY \leftarrow -\infty$ ; $(x, y, k) \leftarrow (NIL, NIL, NIL)$ ;
2: **Pour tout** $(x_{Cour}, y_{Cour}, k_{Cour})$ dans $LibreXY(i_0)$ **Faire**
3: insérer $i_0$ dans $\Gamma$ suivant les paramètres $(x_{Cour}, y_{Cour}, k_{Cour})$ ; $minINSER_{Cour} \leftarrow +\infty$ ;
4: **Pour tout** $i \in$ *D1*-$\{i_0\}$ **Faire**
5: $tempINSER \leftarrow INSER(i)$ ;
6: **Si** $tempINSER < minINSER_{Cour}$ **Alors**
7: $minINSER_{Cour} \leftarrow tempINSER$ ;
8: **Fin si**
9: **Fin pour**
10: **Si** $minINSER_{Cour} > maxMinINSERXY$ **Alors**
11: $(x, y, k) \leftarrow (x_{Cour}, y_{Cour}, k_{Cour})$ ; $maxMinINSERXY \leftarrow minINSER_{Cour}$ ;
12: **Fin si**
13: retirer $i_0$ de $\Gamma$ ;
14: **Fin pour**
15: **Retourner** $(x, y, k)$ ;

Nous remarquons rapidement que l'application d'un tel algorithme est particulièrement chronophage et qu'il est important d'unifier les différents calculs quitte à travailler sur des valeurs approchées.

On suppose alors que, chaque fois que l'insertion d'une demande cible $i_0$ est envisagée, on dispose, pour toute demande $i$ non encore intégrée, de chaque mesure d'*Insérabilité* $INSER2(i, \Gamma_k, x, y)$, $INSER1(i, \Gamma_k)$ et $INSER(i, \Gamma)$, $k \in NbCarLibre(i)$, $(x, y) \in LibreXY(i)$. L'insertion de $i_0$ dans $\Gamma_k$ selon le couple d'insertion $(x, y)$ étant sur le point d'être testée, on calcule, pour chaque $i \neq i_0$ et non encore insérée, la quantité $H(i)$ donnée en formule 6.1.

$$H(i) = INSER(i, \Gamma) - INSER1(i, \Gamma_k) \tag{6.1}$$

Pour chaque demande $i$ dans $D1$-$\{i_0\}$, tel que $k \in NbCarLibre(i)$, on teste alors l'insertion de $i$ dans $Inserted(\Gamma, i_0, k, x, y)_k$ (correspondant à la tournée $\Gamma_k$ après insertion de $i_0$), en utilisant, au niveau de la boucle de contrôle pilotant les couples d'insertion possible, le même schéma que celui de la mise à jour des quantités $NbCarLibre(i)$ et $INSER(i, k)$. On mémorise les quantités $INSER1(i, Inserted(\Gamma, i_0, k, x, y))$ découlant de ce processus. On déduit, pour chaque $i$ ainsi testée, la quantité $K(i) = INSER(i, Inserted(\Gamma, i_0, k, x, y)) = H(i)+INSER1(i, Inserted(\Gamma, i_0, k, x, y)_k)$, et donc, *In Fine*, la quantité $Val(k, x, y) = Min_{i \in D1-i_0}\ (K(i) + H(i))$. La solution de notre problème est donc le triplé $(k,\ x,\ y)$ qui induit la plus grande valeur $Val(k, x, y)$. Il est à noter que, à l'issue de ce processus, la mise à jour des quantités de mesure d'*Insérabilité* $INSER2(i, \Gamma_k, x, y)$, $INSER1(i, \Gamma_k)$ et $INSER(i, \Gamma)$, $k \in NbCarLibre(i)$, $(x, y) \in LibreXY(i)_k$, se fait de façon immédiate, puisque toutes ces valeurs ont déjà été calculées en cours de processus.

A l'itération $r$, la sélection des paramètres d'insertion $(x_r, y_r, k_r)$ de la demande sélectionnée $i_r$ se fait sur la maximisation de la plus petite valeur d'*Insérabilité* des demandes restantes, soit de $D1-i_r$. Il convient donc de choisir $i_{r+1}$ selon l'*Insérabilité* la plus réduite puisque l'insertion précédente a été réalisée dans le but de la maximiser. Autrement dit, si le processus maximise $Min_{i \in D1-i_r} INSER(i,\ Inserted(\Gamma,\ i_r,\ k,\ x,\ y))$ de manière exacte, il est souhaitable que la demande $i_{r+1}$ soit celle pour laquelle on a choisi $(x_r, y_r, k_r)$ de façon à maximiser son *Insérabilité*. Ceci étant, on remarque que, une fois la toute première demande sélectionnée, le processus devient déterministe. Afin d'introduire du non-déterminisme, nous injectons de l'incertitude en maximisant la somme des valeurs *INSER* des $n$ demandes aux plus petites *Insérabilités* et, au moment de leur sélection à l'itération suivante, nous choisissons parmi ces $n$ requêtes de manière aléatoire.

**Remarques.** La remise à jour des quantités $NbCarLibre(i)$ et $LibreXY(i)_k$ est à priori inutile si l'on suit le schéma ci-dessus à la lettre, puisqu'elle est incluse dans les calculs de mesure d'*Insérabilité*. En fait, comme on va le voir à présent, il peut être utile de conserver cette phase de remise à jour telle qu'on l'a pratiquée jusqu'à présent. En effet, le nombre de tests d'insertion de $i$ dans $Inserted(\Gamma, i_0, k, x, y)_k$ pouvant être ici sensiblement plus élevé que celui effectué une fois que les paramètres $k$, $x$, $y$ ont été fixés, il est souhaitable d'introduire des procédés de filtrage et d'approximation des quantités $INSER2(i, Inserted(\Gamma, i_0, k, x, y)_k, x', y')$, qui permettent une accélération du processus général d'insertion de $i_0$.

Ce qui précède met l'accent sur la seule recherche de solutions réalisables. Dans le cas général où une fonction Objectif est présente, la démarche qui vient d'être décrite s'adapte aisément. Nous choisissons les paramètres $k$, $x$, $y$ en minimisant $Eval(Inserted(\Gamma, i_0, k, x, y), t) - \lambda Min_{i \in D1\text{-}\{i_0\}} INSER(i, Inserted(\Gamma, i_0, k, x, y))$, dans laquelle le coefficient $\lambda$ pondère l'*Insérabilité* par rapport à la quantité $Eval(\Gamma, t)$.

**Filtrage.** Le terme filtrage désigne ici le fait d'éviter des opérations inutiles. Considérons ici une situation où l'insertion d'une cible $i_0$ est testée dans un véhicule $k$, selon le couple d'insertion $(x, y)$. On va donc balayer l'ensemble des demandes $i$ dans $D1 - \{i_0\}$, telles que $k \in NbCarLibre(i)$.

Soit $W_1$ la mesure $Min_{i \in \{D1\text{-}i_0\}} INSER(i, \Gamma)$ et $H(i)$ la quantité $H(i) = INSER(i, \Gamma) - INSER1(i, \Gamma_k)$. Si $H(i) \geq W_1$ alors il est inutile de tester l'impact sur $i$ de l'insertion de $i_0$ dans $\Gamma_k$ selon le couple $(x, y)$.

Ce mécanisme peut être étendu au fur et à mesure que des demandes $i$ sont testées. Soit, à un instant donné, $D2$ l'ensemble des demandes $i$ dans $D1 - \{i_0\}$, telles que $k \in NbCarLibre(i)$, et qui ont été testées au sens de l'impact de l'insertion de $i_0$ dans $\Gamma_k$ selon le couple *(x,y)* sur leur *Insérabilité* dans $\Gamma_k$, et soit $W_2$ la mesure $Min_{i \in D2} INSER(i, Inserted(\Gamma, i_0, k, x, y))$. Si $H(i) \geq W_2$ alors il est inutile de tester l'impact sur $i$ de l'insertion de $i_0$ dans $\Gamma_k$ selon le couple $(x, y)$.

**Approximation.** Le calcul exact des quantités $INSER1(i, Inserted(\Gamma, i_0, k, x, y)_k)$ est potentiellement coûteux en temps, si l'on prend en compte le fait qu'il a à être exécuté un nombre de fois assez important. On peut donc chercher à l'alléger en le remplaçant par un calcul approché plus simple.

Soit donc une situation où l'on teste l'insertion d'une demande cible $i_0$ dans $\Gamma_k$ selon le couple $(x,y)$. La mise en œuvre de ce test suppose un calcul (exact ou approché) des fenêtres de temps associées à $o_{i_0}$ et $d_{i_0}$, ainsi qu'à $DepotD_k$ et $DepotA_k$. Elle permet de dériver une tournée réduite $\Gamma_{kr}$, correspondant au parcours $(DepotD_{kr}, o_{i_0}, d_{i_0}, DepotA_{kr})$ et aux fenêtres de temps de ces 4 nœuds. La charge sur cette tournée se répartit naturellement, étant nulle sur les deux extrémités et égale à la charge minimale de $Inserted(\Gamma, i_0, k, x, y)_k$ entre $o_{i_0}$ et $d_{i_0}$. On se contente alors d'évaluer $INSER1(i, \Gamma_{kr})$ et de considérer que $INSER1(i,\ Inserted(\Gamma, i_0, k, x, y)_k)$ est égal à $Min(INSER1(i, \Gamma_{kr}), INSER1(i, \Gamma_k))$.

Cette approximation ne vaut que dans la phase de test de l'insertion de $i_0$ dans dans $\Gamma_k$ selon le couple $(x, y)$. Au moment où cette insertion est réellement effectuée, c'est-à-dire au moment où les données $NbCarLibre(i)$ et $LibreXY(i)_k$ sont remises à jour, les quantités $INSER2(i, \Gamma_k, x, y)$ et $INSER1(i, \Gamma_k)$ sont évaluées de façon exacte.

### 6.1.3 Prise en compte de rejets de demandes

Tout ce qui a été développé précédemment se situe en continuité de la version du DARP telle qu'elle a été étudiée dans les chapitres antérieurs. Satisfaire l'ensemble des demandes y est considéré comme une contrainte dure. Le processus d'insertion est relancé jusqu'à ce que l'on obtienne au moins une fois une solution réalisant toutes les demandes. On pilote donc le choix de la demande cible $i_0$ de façon à minimiser le risque qu'une demande s'avère à un moment donné impossible à satisfaire. Dans ce cadre, les sections précédentes sur la notion d'*Insérabilité* tendent en premier lieu à contribuer à minimiser le risque d'échec : à chaque tentative d'insertion d'une demande cible $i_0$, on va rechercher les paramètres d'insertion

$(k, x, y)$ de façon à minimiser une quantité globale : $Eval(Inserted(\Gamma, i_0, k, x, y), t) - \lambda Min_{i \in D1\text{-}\{i_0\}} INSER(i, Inserted(\Gamma, i_0, k, x, y))$, $D1$ désignant ici l'ensemble des demandes non traitées, $\lambda$ désignant un coefficient positif et $Eval$ désignant la mesure de qualité standard utilisée dans les autres chapitres.

Si l'on accepte à présent l'idée que certaines demandes pourront ne pas être satisfaites, et que la performance globale de l'ensemble des tournées construites $\Gamma$ va s'exprimer sous la forme : $COUT(\Gamma, t) = Eval(\Gamma, t) - \mu.|D\text{-}Rejet|$, où $D\text{-}Rejet$ désigne l'ensemble des demandes qui n'ont pu être traitées et où $\mu$ est un coefficient positif, alors on est amené à prendre en compte le fait qu'insérer à tout prix la demande cible peut être négatif pour l'insertion des autres demandes. Deux points doivent dès lors être réexaminés :

— il faut reprendre le critère qui induit le choix de $i_0$,
— $i_0$ étant la demande cible courante, il faut reprendre le processus qui décide des paramètres d'insertion, et qui, éventuellement, conclut à la non insertion.

#### Modification du processus de choix de la demande cible à insérer

Dans le processus courant, le choix de $i_0$ est basé sur une règle de priorité avantageant les demandes $i$ pour lesquelles l'*Insérabilité* $INSER(i, \Gamma)$ ou bien $NbCarLibre$ de $i$ est faible. Un tel choix peut être adapté à une situation intégrant l'hypothèse de rejets, en utilisant le fait que le processus d'insertions successives, basé sur un balayage des demandes de $D$ dans un ordre partiellement aléatoire, est en principe lancé plusieurs fois (un nombre $N$ de fois, où $N$ est le nombre de réplications). On pénalise donc, au fur et à mesure que le processus avance, chaque demande insérée $i_0$ dont l'insertion occasionne une forte dégradation mesurée par la quantité $\sum_{i \in D1\text{-}\{i_0\}}(INSER(i, \Gamma)) - INSER(i, Inserted(\Gamma, i_0, k, x, y))$. A chaque exécution du processus de balayage de $D$, la demande $i_0$ est alors choisie sur une base de priorité combinant les quantités $INSER(i_0, \Gamma)$ et $PENALITE(i_0)$ : $i_0$ est choisi aléatoirement parmi les 2 ou 3 demandes $i$ minimisant $\Lambda(INSER(i_0, \Gamma), PENALITE(i_0))$ où $\Lambda$ est une fonction positive croissante.

#### Calcul des paramètres d'insertion, ou de la décision de refus d'insertion

La demande $i_0$ à insérer ayant été choisie, nous devons décider ici, soit des paramètres d'insertion $k$, $x$, $y$, soit du rejet de la demande $i_0$. Nous envisageons pour cela 4 approches :

La **première approche** consiste à insérer coûte que coûte $i_0$ dès lors que c'est possible, en utilisant le critère utilisé dans la section précédente. Clairement, dans ce cas, $i_0$ ne sera rejetée qu'au cas où il ne serait pas insérable.

La **deuxième approche** consiste à insérer coûte que coûte $i_0$ dès lors que c'est possible, en modifiant le critère utilisé. On choisit dès lors $(k, x, y)$ (s'ils existent) de façon à minimiser une quantité : $Eval(Inserted(\Gamma, i_0, k, x, y), t) - \mu. \sum_{i \in D1\text{-}\{i_0\}} \Phi($ $INSER(i, Inserted(\Gamma, i_0, k, x, y)))$, où $\Phi$ est une fonction de $\mathbb{R}^+$ dans $[0, 1]$, croissante et dont le profil soit du type de la fonction $1 - e^{-x}$. Cette formulation atténue

l'influence des demandes dont l'insertion paraît le moins probable. Clairement, dans ce cas, $i_0$ ne sera rejetée que si elle n'est pas insérable.

La **troisième approche** consiste à insérer coûte que coûte $i_0$ et à choisir $(k, x, y)$ de façon à minimiser $Eval(Inserted(\Gamma, i_0, k, x, y), t) - \lambda.Min_{i \in D1\text{-}\{i_0\} - D\text{-}Rejet}\ INSER(i, Inserted(\Gamma, i_0, k, x, y)) + \mu.|D\text{-}Rejet|)$, $D\text{-}Rejet$ étant l'ensemble des demandes de $D1$ pour lesquelles la quantité $INSER(i, Inserted(\Gamma, i_0, k, x, y))$ est nulle (demandes qui vont à priori devenir non insérables). Il est clair que, dans ce dernier cas, $i_0$ ne sera rejetée que si elle n'est pas insérable.

La **quatrième approche** admet l'idée d'un rejet de $i_0$ même si $i_0$ est insérable. On décide de rejeter $i_0$ si le nombre de demandes dont l'*Insérabilité* se dégrade, du fait de l'insertion de $i_0$, est important. On calcule alors $k$, $x$, $y$ qui minimisent la quantité $Eval(Inserted(\Gamma, i_0, k, x, y), t) - \mu. \sum_{i \in D1\text{-}\{i_0\}} \Phi\ (INSER(i, Inserted(\Gamma, i_0, k, x, y)))$, de la deuxième approche ci-dessus, et refuse l'insertion de $i_0$ dans le cas où la différence $\sum_{i \in D1 - \{i_0\}}(\Phi(INSER(i, \Gamma))) - \Phi(INSER\ (i, Inserted\ (\Gamma, i_0, k, x, y)))$ est supérieure à une certaine quantité *SEUIL*. Cette quantité est évolutive en cours du processus. Elle décroît de façon à renforcer les chances d'insérer chaque demande.

**Remarque.** Pour la quatrième approche, on pourrait envisager de pondérer les demandes de telle sorte que les demandes $i$ à fortes charges $Q_i$ et longs acheminements DIST($o_i$, $d_i$) ne soient pas systématiquement rejetées.

### 6.1.4 Résultats expérimentaux

**Résumé**

Nous cherchons ici à évaluer l'apport de la notion d'*Insérabilité* en tant qu'outil d'anticipation au sein du processus d'insertion pour le DARP statique. Pour ce faire, nous nous focalisons sur :

— la sélection de la demande. Nous évaluons, pour une instance difficile qui requiert l'intégration de toutes les demandes, l'apport de l'*Insérabilité* aux techniques présentées dans le chapitre 3. Au passage, nous analysons l'évolution de la valeur $INSER$ tout au long d'une réplication de l'algorithme.

— la sélection des paramètres d'insertion de la demande cible. Nous regardons ici l'intérêt de la mesure d'*Insérabilité* sur des instances "maison". Ici, nous autorisons l'exclusion de demandes, ce qui nous permet d'étudier également l'utilisation de la mesure pour cerner les demandes qui sont à exclure.

**Environnement et méthodes implémentées.** Les méthodes développées ici le sont à nouveau en C++ et exécutées sur un système Linux 64 bits. Nous implémentons les processus d'insertions successives proposés en 6.1.2 et 6.1.3. Dans le cas avec rejets, ce sont les approches 3 et 4 qui sont développées.

**$1^{er}$ Test : Apport de la mesure d'*Insérabilité* à la sélection de la demande cible**

Nous utilisons ici les instances de [Cordeau and Laporte (2003)] qui voyaient le processus d'insertion échouer, pour certaines, sur une grande partie des réplications. En effet, pour ces réplications, une partie des demandes ne pouvait plus être satisfaite. Ici, nous ne modifions le processus heuristique de résolution décrit dans le chapitre qu'au moment de la sélection de la demande. Le calcul des paramètres d'insertion de celle-ci reste inchangé tout comme la pondération des différents critères de la fonction Objectif dont le calcul reste lui aussi constant.

Les instances [Cordeau and Laporte (2003)] sont au nombre de 20 et sont particulièrement difficiles à traiter au niveau des contraintes de temps. $Taux_{Succes}^{DARP}$ et $Taux_{Succes}^{DARPRob}$ du tableau 6.1 de la page suivante sont les taux de réussite des 100 réplications obtenues respectivement par la résolution heuristique du chapitre 3, et pour l'approche décrite en 6.1.2.

Bien que les taux de succès soient bons dans la majorité des instances, nous remarquons que les instances pr08, pr09, pr19, pr20 et surtout pr10 demeurent difficiles. La prise en compte de la notion d'*Insérabilité*, induit une nette amélioration sans pour autant que soit obtenu un succès à chaque réplication.

| Inst. | $Taux_{Succes}^{DARP}$ | $Taux_{Succes}^{DARPRob}$ |
|---|---|---|
| pr01 | 99 | 100 |
| pr02 | 100 | 100 |
| pr03 | 97 | 100 |
| pr04 | 100 | 100 |
| pr05 | 100 | 100 |
| pr06 | 100 | 100 |
| pr07 | 90 | 96 |
| pr08 | 56 | **91** |
| pr09 | 18 | 21 |
| pr10 | 1 | **7** |
| pr11 | 100 | 100 |
| pr12 | 100 | 100 |
| pr13 | 99 | 100 |
| pr14 | 100 | 100 |
| pr15 | 100 | 100 |
| pr16 | 100 | 100 |
| pr17 | 98 | 100 |
| pr18 | 99 | 100 |
| pr19 | 64 | **99** |
| pr20 | 43 | 56 |
| Moy. | 83,2 | 88.5 |

TABLE 6.1 – Taux de succès INSERTION / INSERTION avec *Insérabilité*

**Evaluation en cours de processus de la mesure d'*Insérabilité*.** La figure 6.2 décrit l'évolution de la mesure d'*Insérabilité INSER* au cours d'une exécution du processus de résolution sur l'instance la plus difficile de [Cordeau and Laporte (2003)] : la pr10, pour laquelle $Taux_{Succes}^{DARPRob}$ est égal à 7%.

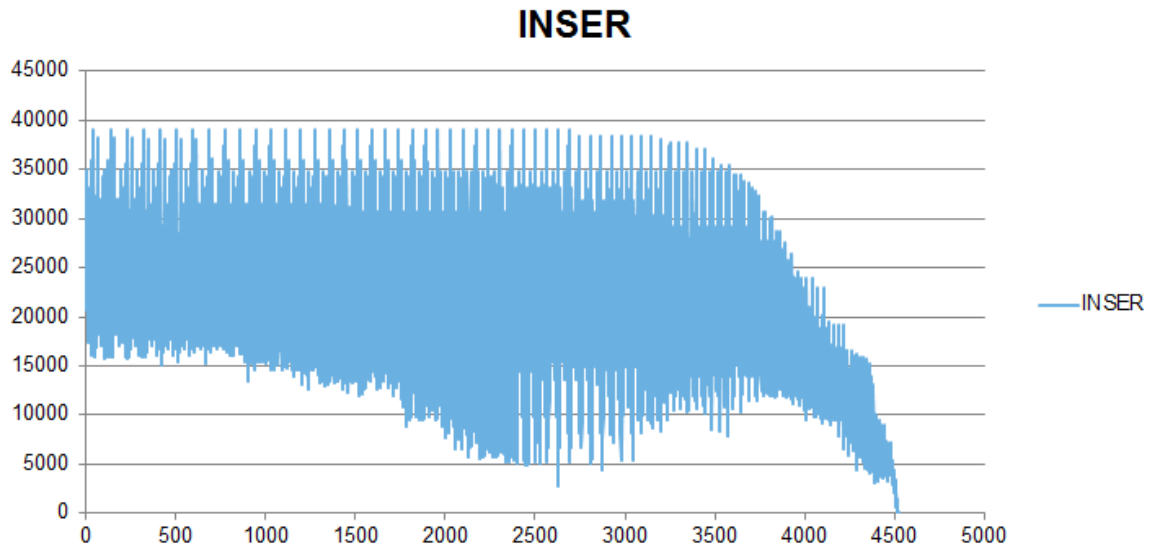


FIGURE 6.2 – Relevé de la valeur *INSER* le long de la résolution de pr10

Plus de 4500 valeurs ont été relevées tout au long du processus tentant d'insérer les 144 demandes. Les valeurs correspondent bien entendu aux demandes restant à insérer et leur propre mesure d'*Insérabilité* varie au fil des insertions des autres demandes. L'abscisse de ce graphique désigne l'index (l'instant en cours de processus) où une valeur *INSER* est calculée.

En début de processus, le nombre de demandes à insérer est important d'où le grand éventail de valeurs pour *INSER* (de 15000 à 40000). En cours de processus, les *INSER* subissent de fortes baisses pour certaines demandes qui sont alors sélectionnées et insérées (d'où une remontée des valeurs les plus basses à partir du relevé 3000). En fin de processus, les demandes restantes subissent une forte dégradation de leur valeur *INSER* suite aux insertions déjà réalisées. Nous obtenons naturellement une valeur nulle en fin de processus, qui a vu le rejet d'une demande.

#### 2ème Test : Apport de la mesure d'*Insérabilité* à la sélection des paramètres d'insertion

Nous travaillons ici dans la perspective d'insérer le plus possible de demandes sans qu'il soit possible pour autant de le faire pour toutes. Nous testons ici trois heuristiques :

1. l'heuristique d'insertions aléatoires du chapitre 3,
2. l'heuristique d'insertions aléatoires de la section 6.1.2,
3. l'heuristique de la section 6.1.3, qui induit l'exclusion des demandes dont la prise en charge implique une trop grande variation de l'*Insérabilité* des demandes restantes (4ème approche).

Dans chaque cas, la sélection de la demande est effectuée sur la base des plus petites cardinalités de *NbCarLibre*.

**Analyse du taux de rejet induit par les heuristiques 1 et 2.** Nous générons les instances selon les caractéristiques du tableau 6.2, $EF(o)$ et $EF(d)$, désignant respectivement les amplitudes des fenêtres de temps à l'origine et à la destination, communes à toutes les demandes de $D$.

| K | $EF(o)$ | $EF(d)$ | $\Delta$ | $CAP$ |
|---|---|---|---|---|
| 10 | 35 | 10 | $\infty$ | 10 |

TABLE 6.2 – Paramètres du processus de génération d'instances

Le tableau 6.3 de la page suivante fournit alors les taux d'insertion moyens obtenus respectivement avec 10 réplications des heuristiques 1 et 2, dès lors que $|D|$ varie, pour des groupes de 5 instances chacun.

| $\|D\|$ | 50 | 75 | 100 | 150 | 200 |
|---|---|---|---|---|---|
| $T_{Insert}$ | 100 | 93.2 | 78.9 | 64.2 | 52.6 |
| $T_{Insert_{Rob}}$ | 100 | 96.8 | 85.3 | 66.4 | 54.1 |
| $Gap_{Insert}$ | 0 | 3.86 | 8.11 | 3.43 | 2.81 |

TABLE 6.3 – Gap entre les taux *INSER*

Nous obtenons une augmentation qui peut aller jusqu'à 8% de demandes insérées en plus grâce à la sélection des paramètres ($x$, $y$, $k$) selon la variation de l'*Insérabilité*.

**Analyse du comportement de l'heuristique 3.** La 4ème approche de l'heuristique de la section 6.1.3 autorise le rejet d'une demande dès lors qu'une quantité $\sum_{i \in D1\text{-}\{i_0\}}(\Phi(INSER(i,\Gamma))) - \Phi(INSER(i, Inserted(\Gamma, i_0, k, x, y)))$ excède une valeur *SEUIL*, ($x$, $y$, $k$) désignant ici les meilleurs paramètres d'insertion induits. Nous testons ici la sensibilité du processus à cette valeur $SEUIL$.

La figure 6.3 décrit une évolution des quantités $INSERav = \sum_{i \in D1\text{-}\{i_0\}} \Phi(INSER(i,\Gamma))$ et $INSERap = \sum_{i \in D1\text{-}\{i_0\}} \Phi(INSER(i, Inserted(\Gamma, i_0, k, x, y)))$ et leur différence *Variation*.

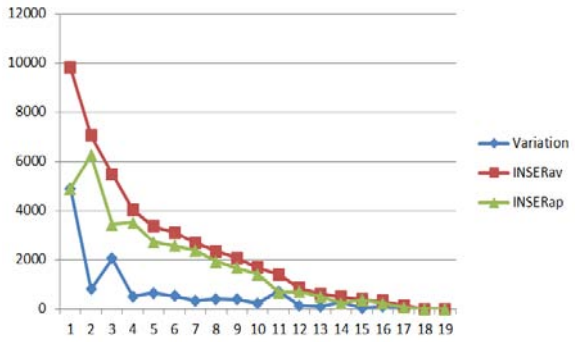


FIGURE 6.3 – Variation des valeurs *INSER*

Ce diagramme montre qu'il est nécessaire d'avoir une valeur *SEUIL* dynamique dans le sens où elle doit décroître au fil de l'exécution.

En appliquant maintenant notre heuristique 3, aux instances à 100 demandes et 10 véhicules, nous constatons un taux moyen $T_{Insert_{Rob}}$ de 86,6% au lieu des 85,3% du tableau 6.3, ce qui constitue un gain relativement modeste.

## 6.2 Robustesse dans le DARP dynamique

Cette section traite du *Dial-a-Ride* dynamique, via l'adaptation des procédures d'insertions du chapitre 3, et en utilisant les concepts développés en 6.1 afin d'apporter de la robustesse au système. Le mot robustesse renvoie ici à la possibilité du système d'insérer les demandes à venir, demandes non encore émises par les usagers. Nous avons donc besoin d'une représentation de ces demandes (nous les appellerons *demandes virtuelles*) afin de les anticiper. Cette représentation sera considérée ici comme connue. Son acquisition dépend du contexte, et le travail de statistiques nécessaire à son instanciation sort largement du cadre de cette thèse. Décrivons d'abord le fonctionnement du système "DARP dynamique" tel que nous l'envisageons ici.

Soit $t_a$ une date à laquelle le processus d'allocation des véhicules aux demandes, appelé ROUTE, devra être exécuté. Un premier module (DEC) analyse le système afin de déterminer $t_a$. L'état du système est alors donné par le réseau courant et un ensemble $D1_a$ des demandes émises par les usagers mais non encore traitées ni insérées dans la collection $\Gamma_a$ des tournées courantes. Ces tournées comportent un ensemble de demandes à satisfaire ou en cours de satisfaction mais qui ont toutes déjà été insérées par de précédentes exécutions de ROUTE.

Au moment $t_a$, ROUTE crée alors une représentation des probables demandes futures. Ces *demandes virtuelles* forment l'ensemble $DV_a$. $\Gamma_a$, $D1_a$ et $DV_a$ deviennent alors les entrées du sous-processus MAIN-ROUTE qui, en sortie, fournit les nouvelles tournées $\Gamma_b$ accompagnées des nouvelles dates de passage (cf. résumé de l'ensemble du processus en figure 6.4).

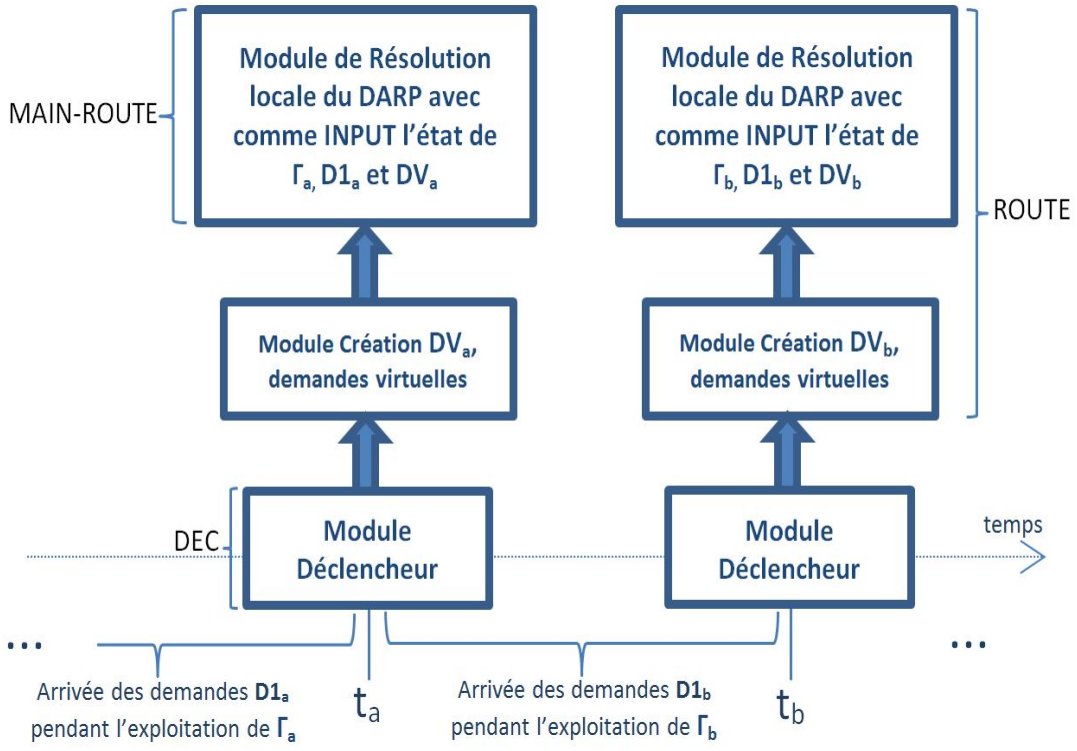


FIGURE 6.4 – Principaux processus autour des résolutions locales dans le DARP dynamique

MAIN-ROUTE dispose alors de l'ensemble des informations permettant de rendre le système plus robuste grâce à la prise en compte des demandes virtuelles auxquelles sont appliquées les différentes utilisations de la mesure d'*Insérabilité* de la section précédente. La réponse de MAIN-ROUTE aux différentes demandes inclut alors non seulement un message d'acceptation ou de rejet, mais aussi un "rendez-vous", dont la date doit être fixée de façon à laisser suffisamment d'espace pour les demandes futures.

### 6.2.1 La demande virtuelle $DV^*$

Nous qualifions de *demandes virtuelles* l'ensemble $DV$ des demandes à venir telles que le système les perçoit. On représente la *demande virtuelle* $DV^*$ sous une forme probabiliste, selon l'écriture suivante :

— $DV^* = \sum_{i \in DV} p_i.D_i$, où les $D_i, i \in DV$, sont des demandes de la même forme que les demandes réelles. Elles sont donc représentées sous forme de sextuplets (origine = $o_i$, destination = $d_i$, charge = $Q_i$, $F(o_i)$ et $F(d_i)$ = fenêtres de temps sur l'origine et la destination, $\Delta_i$ = limite sur la durée d'acheminement). Les $p_i$ sont des coefficients positifs.

Les coefficients $p_i$ ne sont pas des probabilités, mais des espérances : formellement, $p_i$ peut être vu comme l'espérance du nombre de fois où la demande $i$ sera émise durant le futur envisagé. Ceci a pour implication que la notation $p_i.D_i$ ne renvoie à aucune notion de produit d'une demande par un scalaire, mais simplement à une notion d'itération abstraite.

#### Génération des demandes virtuelles

On n'évoquera pas ici en détail de ce que peut être le processus de génération effective d'une telle représentation. Il dépend à priori du contexte et de l'information statistique dont on dispose initialement sur les demandes à venir. Ce processus sera le plus souvent basé sur une notion d'échantillonnage. En revanche, deux points importants doivent être évoqués :

— telle que justifiée au paragraphe précédent, la représentation $DV$ est susceptible d'impliquer un ensemble de taille très importante. Or, l'utilisation qui en sera faite supposera que le nombre de demandes soit réduit (au plus une centaine d'éléments). La conséquence en est que la génération de $DV^*$ devra induire un processus d'agrégation, tant du réseau à l'intérieur duquel les véhicules circulent que de l'espace temps, de façon à permettre un regroupement des demandes générées par échantillonnage en un nombre limité de demandes ;

— dans la mesure où l'introduction de $DV^*$ est liée au contexte dynamique, ce processus de génération de $DV^*$ devra pouvoir être activé à tout instant t, pour un horizon [t, t + $\eta$] donné, considéré comme l'horizon utile pour une bonne gestion des demandes émises avant l'instant t et encore en attente de traitement. Dans la mesure où l'exécution de ce processus à cet instant t sera soumise à de fortes contraintes de temps, il sera souhaitable qu'ait eu lieu un

prétraitement qui génère une représentation $DV^*$ sur un horizon plus complet [Début, Fin] incluant l'intervalle [t, t + $\eta$], de telle sorte que la partie associée à [t, t + $\eta$] soit accessible de façon directe, sans qu'aucun recours à un procédé d'échantillonnage ou à un procédé d'agrégation ne soit nécessaire.

**Intégration des demandes virtuelles aux critères d'évaluation**

Plaçons-nous au sein du processus d'insertion et considérons que les données relatives aux demandes non encore insérées (*INSER*, *NbCarLibre(i)*, *LibreXY...*) concerneront ici non seulement $D1$ (l'ensemble de demandes restant à insérer), mais en fait $D1 \cup DV$. Une fois $i_0$ sélectionnée, le critère d'optimisation qui prévaudra pour le calcul des paramètres ($k$, $x$, $y$) d'insertion de $i_0$, portera sur la minimisation d'une quantité donnée par la formule 6.2, *Eval* désignant la méthode de calcul des coûts usuels d'une collection de tournées, et $\Phi$ étant une fonction de normalisation de $\mathbb{R}^+$ dans [0,1], croissante :

$$\begin{aligned} &Eval(Inserted(\Gamma, i_0, k, x, y)) - \sigma. \sum_{i \in D1\text{-}\{i_0\}} \Phi(INSER(i, Inserted(\Gamma, i_0, k, x, y))) \\ &\qquad -\beta. \sum_{i \in DV} p_i.\Phi(INSER(i, Inserted(\Gamma, i_0, k, x, y))) \end{aligned} \tag{6.2}$$

Se pose alors la question de la valeur des coefficients multicritères $\sigma$, $\beta$. Le point clé tient bien sûr ici à ce qu'est la performance globale d'une solution, qui, cette fois-ci, intègre forcément le rejet éventuel de demandes, et s'inscrit dans un contexte d'appels répétés de ROUTE. Pour un appel donné (localement), on fait en sorte que le coût que ROUTE cherche à minimiser s'écrive :

$$COUT(\Gamma, t) = Eval(\Gamma, t) + \mu. |DR\text{-}Rejet| - \beta. \sum_{i \in DV} p_i.\Phi(INSER(i, \Gamma)) \tag{6.3}$$

Le premier élément désigne l'évaluation de la collection de tournées $\Gamma$ au sens des critères usuels, *DR-Rejet* désigne l'ensemble des demandes qui n'ont pu être traitées par ROUTE, et le dernier terme de *COUT* exprime le besoin de garder les demandes virtuelles les plus insérables possible.

On impose $\beta \leq \sigma$ de façon à intégrer le fait que les demandes de $DV$ sont incertaines. En ce qui concerne la relation entre $\sigma$ et $\mu$, on constate qu'un rejet correspond à une perte de valeur normalisée $\Phi(INSER)$ de 1 à 0, et on pose donc $\sigma = \mu$. En fait, on peut voir $\Phi(INSER(i, \Gamma))$ comme l'espérance du booléen : "$i$ sera insérée".

En ce qui concerne enfin les ratios (donc égaux) $\beta/\sigma$ et $\beta/\mu$, il constitue alors un paramètre, conditionné par la confiance que l'on met sur le modèle de représentation des demandes virtuelles. En première approche, sa valeur de compromis est fixée à 1/2.

### 6.2.2 Le modèle "DARP local" - MAIN-ROUTE

Le modèle considère, à l'instant $T$, les **INPUT** suivants :

- l'ensemble $DR$ des demandes réelles apparues depuis le précédent appel à MAIN-ROUTE,
- la demande virtuelle $DV^* = \sum_{i \in DV} p_i.D_i$, générée à partir de $T$,
- l'ensemble $\Gamma = (\Gamma_k, k = 1..K)$, des tournées associées à un ensemble $D0$ de demandes déjà insérées. Ces tournées sont datées par une fonction $t$ et, pour chaque demande $i \in D0$, $t(o_i)$ constitue la date de "rendez-vous" pour $i$, qui doit être préservée. Les nœuds $DepotD_k$ et $DepotA_k$ font partie de chaque tournée $\Gamma = (\Gamma_k, k = 1..K)$.

MAIN-ROUTE doit produire les **OUTPUT** suivants :

- un sous-ensemble $DR\text{-}Rejet$ de demandes rejetées,
- l'ensemble $\Gamma = (\Gamma_k, k = 1..K)$, auquel sont intégrées les demandes de $DR - DR\text{-}Rejet$,
- une fonction date $t$, qui date $\Gamma$, et qui doit être telle que, pour chaque $i \in D0$, $t(o_i)$ est demeurée identique à ce qu'elle était en INPUT. Pour tout $i \in DR - DR\text{-}Rejet$, $t(o_i)$ est dite "date de rendez-vous" assignée à $i$,
- la valeur $COUT(\Gamma, t)$ induite, calculée selon la formule 6.3 : $COUT(\Gamma, t) = Eval(\Gamma, t) + \mu.|DR\text{-}Rejet| - \beta.\sum_{i \in DV} p_i.\Phi(INSER(i, \Gamma))$.

### 6.2.3 Processus d'insertion

Il découle de ce qui précède que le processus d'insertion des demandes d'un ensemble $DR$ de demandes "réelles" avec prise en compte de la notion de robustesse (c'est-à-dire avec prise en compte de l'impact de l'insertion courante sur la demande aléatoire $DV$) se fait en considérant que l'on a à insérer les demandes de $DR$, en préservant autant que faire se peut, à la fois l'*Insérabilité* des demandes de $DR$ restant à insérer et celles des demandes de $DV = \{D_i,\ i \in I\}$. A chaque itération du processus d'insertion, nous retrouvons les deux étapes clés du cas standard :

- la sélection de la demande $i_0$ à insérer,
- le choix des paramètres d'insertion ($k$, $x$, $y$) pour insérer $i_0$ et qui minimisent $COUT$.

Les 2 étapes se gèrent selon les processus vus en 6.1.3, la formule 6.3 fournissant le critère auxiliaire utilisé pour piloter le choix effectif de $i_0$ et des paramètres $k$, $x$, $y$.

Nous devons maintenant définir la manière dont les rendez-vous synchronisant véhicules et usagers doivent être calculés et ceci toujours dans l'optique de faciliter les futures insertions des demandes à venir.

### 6.2.4 Le problème des rendez-vous

La gestion d'un DARP en contexte dynamique suppose, dans la plupart des cas, que le traitement des demandes débouche sur des rendez-vous fixés par l'opérateur aux demandeurs, c'est-à-dire par l'émission d'une date de service pour le chargement des usagers. Ces derniers ne pourront pas s'attendre à ce que le véhicule soit en avance au point de départ, mais seront sûrs en revanche qu'il n'aura, éventuellement, qu'une petite marge de retard. Ceci explique que, dans notre modèle DARP local, nous imposions que les dates $t(o_i)$, $i \in D0$, ne soient pas modifiées.

La façon dont ces dates de rendez-vous $t(o_i), i \in D0$, sont calculées, est susceptible d'avoir un fort impact sur l'insertion ou la non insertion des demandes à venir.

#### Exemple

La figure 6.5 représente une flotte de 2 véhicules ayant déjà intégré 5 demandes et fixé des rendez-vous aux individus correspondants. L'objectif est ici celui de minimiser les distances totales parcourues par les véhicules. L'algorithme qui contruit ces tournées recherche à ne minimiser que les durées des deux tournées. Pour les 5 premières demandes aux rendez-vous fixés, il s'avère que les contraintes de temps et de capacité permettent de séparer strictement chaque nœud par la distance (temps) qui les sépare.

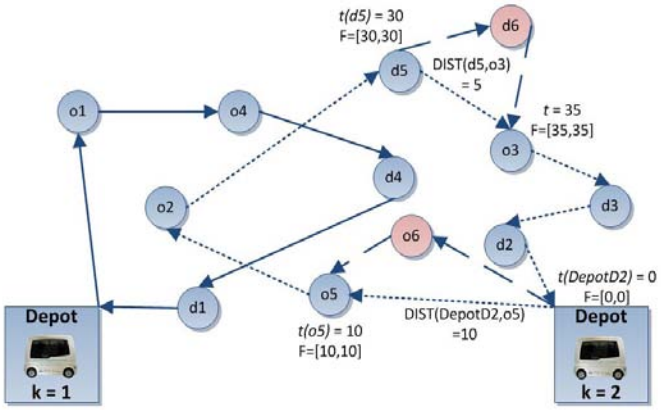


FIGURE 6.5 – Flexibilité des tournées suite aux premiers rendez-vous

Dans un second processus de résolution, la demande $i6$ élargit le graphe de 2 nœuds supplémentaires. Pour chacun de ces nœuds nous avons l'inégalité $Dist(DepotD2, o6) + Dist(o6, o5) > Dist(DepotD2, o5)$ alors que nous avons, suite au calcul des dates de rendez-vous, l'égalité sur les dates de passages : $t(DepotD2) + Dist(DepotD2, o5) = t(o5)$. La demande $i6$ ne pourra donc être insérée au sein des ces tournées sauf si cette dernière se positionne sur le même axe, sans temps de service, ou tout à la fin des plannings de k1 ou k2 si les contraintes de temps le permettent. Cette situation est évidemment très contraignante, il faut donc essayer d'anticiper les futures demandes au moment de fixer les dates de passage des véhicules.

**Formalisation**

Afin de formaliser le problème que ce constat pose, reprenons les données de MAIN-ROUTE, soit l'ensemble des demandes réelles $DR$, la demande aléatoire $DV^*$ $= \sum_{i \in DV} p_i.D_i$, le réseau virtuel $X$ dont les nœuds sont les origines et destinations des demandes de $D = D0 \cup DR \cup DV$ augmentés des nœuds liés au dépôt et les dates de rendez-vous $t(o_i), i \in D0$. Le processus d'insertions successives évoqué en 6.2.3, nous donne :

- un ensemble $\Gamma = (\Gamma_k, k = 1..K)$ de tournées, construites sur les nœuds dérivés de $DR$ ;
- un ensemble $DR\text{-}Rejet \subseteq DR$ de demandes rejetées ;
- pour chaque nœud $x \in X$ figurant dans l'ensemble de tournées $\Gamma$, une fenêtre de temps $F(x)$ : si $x$ s'écrit $o_i$ avec $i \in D0$, alors $F(x)$ est réduite à la date de rendez-vous de $i$ qui n'a pas bougé ;
- pour chaque demande virtuelle $i$ dans $DV$, l'état des fenêtres $F(o_i)$ et $F(d_i)$ qui prend en compte l'existence de $\Gamma$ ;

On doit donc à présent fixer aux demandes $i$ acceptées de l'ensemble $DA = DR - DR\text{-}Rejet$, des dates de rendez-vous t($i$). Cela reviendra, au niveau de la gestion des tournées et des demandes ultérieures, à remplacer les fenêtres $F(o_i)$, $i$ dans $DA$, par des fenêtres de temps [t($i$), t($i$)] d'amplitude nulle, ou, si l'on veut introduire un peu de flexibilité, via une marge de retard et d'avance, par des fenêtres de temps [t($i$) - $\eta$, t($i$) + $\eta$], où $\eta$ est un nombre relativement petit supérieur à zéro. Le critère principal relatif au calcul de ces dates de rendez-vous est le même que le critère global de performance : il s'agit de minimiser le coût des tournées au sens usuel, sans dégrader l'*Insérabilité* des demandes virtuelles de $DV$. Plus précisément, on veut, tout en gardant le fait que les tournées $\Gamma_k$, $k = 1..K$, satisfont les contraintes auxquelles elles sont soumises, minimiser l'expression donnée en formule 6.4.

$$Eval(\Gamma, t) + \alpha. \sum_i p_i.\Phi(INSER(i, \Gamma)) \tag{6.4}$$

**Le Problème simplifié *Mono-Fenêtre***

Le problème ainsi posé est relativement compliqué. On le traite donc de façon heuristique, en s'intéressant d'abord à une version simplifiée, dite Problème *Mono-Fenêtre* :

{ Soit $X \cup \{y_0\}$ un ensemble de variables, soumises à des contraintes linéaires de la forme :

- $y \leq x + A_{x,y}$ (le coefficient peut être égal à $+\infty$) ;
- $x$ doit appartenir à une fenêtre [Min($x$), Max($x$)] ;

On veut donner des valeurs $t(x)$ à chaque $x$ dans $X$, de façon à satisfaire ces contraintes, et à ce que l'amplitude de la fenêtre [u($y_0$,$t$), v($y_0$,$t$)] induite sur $y_0$ par propagation du remplacement de $F(x)$ par $[t(x), t(x)]$ soit la plus grande possible. }

**Interprétation.** On voit que le Problème *Mono-Fenêtre* correspond à notre problème des rendez-vous dans le cas où :

— il existe une seule demande virtuelle, et on mesure son *Insérabilité* à l'aune de l'amplitude de la fenêtre de temps pour sa seule origine ou destination ;

— la quantité $Eval(\Gamma, t)$ n'est pas prise en compte.

On suppose que les fenêtres [Min($x$), Max($x$)], $x$ dans $X \cup \{y_0\}$, sont stables par rapport aux règles d'inférences usuelles associées aux contraintes $y \leq x + A_{x,y}$ : pour tout couple $x$, $y$ ainsi lié par une relation non triviale ($A_{x,y} \neq +\infty$), on a donc :

— $Min(y) \leq Min(x) + A_{x,y}$ ;

— $Max(y) \leq Max(x) + A_{x,y}$.

On peut considérer $X \cup \{y_0\}$ comme muni d'une structure de graphe, selon laquelle chaque arc [$x$, $y$] est associée à une relation $y \leq x + A_{x,y}$, avec $A_{x,y} \neq +\infty$. On suppose bien sûr que ce graphe n'admet pas de circuits négatifs. Il est alors à priori possible (même si ce n'est pas indispensable au niveau algorithmique), de calculer, pour tout couple de nœuds $x$, $y$ dans ce réseau, la longueur $A^*_{x,y}$ d'un plus court chemin de $x$ vers y. On peut dès lors poser, pour tout $x$ dans $X$ : (E1)

— $Max^*(x) = \text{Inf}(Max(x), Max(y_0) + A^*_{y_0,x})$ ;

— $Min^*(x) = \text{Sup}(Min(x), Min(y_0) - A^*_{x,y_0})$.

On vérifie alors assez facilement que :

**Théorème 1.**
On obtient une solution optimale de notre problème en posant $t(x) = \text{Max}^*(x)$ pour tout $x$ dans $X$ (ou $t(x) = \text{Min}^*(x)$). Par ailleurs, les fenêtres [Min*($x$), Max*($x$)], $x$ dans $X$, sont stables entre elles pour les inférences associées aux contraintes $y \leq x + A^*_{x,y}$, et tout vecteur $t = (t(x),\ x \in X)$ qui satisfait ces contraintes induit une solution optimale du Problème Mono-Fenêtre. Au plan algorithmique, la traduction de ce résultat en termes de résolution du Problème *Mono-Fenêtre* vient comme suit :

— on calcule les plus courts chemins, pour les longueurs d'arcs $A_{x,y}$, issus de $y_0$ et arrivant en $y_0$. On obtient donc les quantités $A^*_{y_0,x}$ et $A^*_{x,y_0}$, $x \in X$ ;

— on applique alors les formules (E1), afin de calculer les valeurs $t(x)$, $x \in X$.

#### Le Problème Bi-Fenêtre

On considère maintenant des hypothèses similaires à celles du problème Mono-Fenêtre, à cette différence près que nous disposons à présent de 2 variables cibles $y_0$ et $z_0$, (l'ensemble de variables à considérer et sur lequel sont définies les contraintes binaires $y \leq x + A^*_{x,y}$ est donc l'ensemble $X \cup \{y_0\} \cup \{z_0\}$). On souhaite fixer les valeurs $t(x)$, $x \in X$, de telle sorte que le système global demeure réalisable, et que le produit [-u($y_0$,t)+v($y_0$,t)].[-u($z_0$,t)+v($z_0$,t)] des amplitudes des fenêtres [u( $y_0$,t), v($y_0$,t)] et [u($z_0$,t), v($z_0$,t)] induites par propagation des contraintes et de cette affectation soit le plus grand possible.

**Interprétation.** Le problème correspond à notre problème des rendez-vous dans le cas où l'ensemble $DV$ est réduit à une demande et où $Eval(\Gamma, t)$ n'est pas pris en compte. On vérifie aisément :

**Théorème 2.**
La fonction qui à $t = t(x)$, $x \in X$, associe la quantité [-u($y_0$,t) + v($y_0$,t)].[-u($z_0$,t)+v($z_0$,t)], est concave et linéaire par morceaux.

D'un point de vue théorique, résoudre notre problème Bi-Fenêtre revient donc à maximiser une fonction concave sur un domaine convexe, celui défini par les contraintes portant sur le vecteur $t$ et sur la non vacuité des fenêtres [u($y_0$, t), v($y_0$, t)] et [u($z_0$, t), v($z_0$, t)]. Le contexte dans lequel ce problème se pose (celui de la gestion de la Robustesse dans le DARP) impose toutefois que la solution apportée soit simple et rapide à exécuter, ce qui ne serait pas le cas si l'on cherchait à traiter de ce problème via une méthode de sous-gradient. On se contente dès lors d'appliquer le traitement heuristique 25.

**Algorithme 25** Bi-Fenêtre

1: calculer les quantités $A^*_{y_0,x}$, $A^*_{x,y_0}$, $A^*_{z_0,x}$ et $A^*_{x,z_0}$ avec $x \in X$ ;
2: poser $D^{y_0}_x = A^*_{y_0,x} + A^*_{x,y_0}$ et $D^{z_0}_x = A^*_{z_0,x} + A^*_{x,z_0}$ ;
3: noter $S$ l'ensemble des couples $s = (x, y_0)$ et $s = (x, z_0)$ ; poser alors $D_s = D^{y_0}_x (D^{z_0}_x)$ ; initialiser chaque $D_s$ comme non marqué ;
4: **Tant que** l'ensemble $S$ contient des éléments $s$ non marqués **Faire**
5: identifier $s$ non marqué tel que $D_s$ soit le plus petit possible ;
6: **Si** $s$ est de la forme $(x, y_0)$ **Alors**
7: $Max^*(x) = Inf(Max(x), Max(y_0) + A^*_{y_0,x})$ ;
8: $Min^*(x) = Sup(Min(x), Min(y_0) - A^*_{x,y_0})$ ;
9: $Max(x) = Max^*(x)$ ; $Min(x) = Min^*(x)$ ;
10: propager ce rétrécissement de $F(x)$ sur les autres fenêtres ;
11: **Sinon**
12: procéder de manière symétrique, $s$ étant de la forme $(z_0, x)$ ;
13: **Fin si**
14: **Fin tant que**
15: **Pour tout** $x \in X$ **Faire**
16: $t(x) = Min(x)$ ;
17: **Fin pour**

#### Traitement du Problème des Rendez-vous

Dans ce problème des Rendez-vous, on dispose :
— d'un ensemble $\Gamma = (\Gamma_k, k = 1..K)$ de tournées, construites sur les nœuds de $X$ ;
— pour chaque nœud $x \in X$ figurant dans l'ensemble de tournées $\Gamma$, d'une fenêtre de temps $F(x)$ ;
— pour chaque demande virtuelle $d_i$ dans $DV$, d'un état courant des fenêtres $F(o_i)$ et $F(d_i)$ qui prend en compte l'existence de $\Gamma$.

On veut affecter, à chaque demande $j \in DA$, une date $t(j)$ dans $F(o_j)$, de telle sorte que la quantité : $Eval(\Gamma, t) - \alpha \sum_{i \in DV} p_i \Phi(INSER(i, \Gamma))$, recalculée sur la base de fenêtres de temps $F(o_j)$, $j$ dans $DA$, égales à $[t(j), t(j)]$, soit la plus grande possible. On voit que le cadre est assez proche de celui de la section précédente :

— on dispose d'un ensemble $X$ de variables qui sont associées à chacun des points de passage des tournées de $\Gamma$, ainsi que d'un deuxième ensemble $Y$ associé aux demandes virtuelles $i$, $i \in DV$ ;
— les variables de $X$ sont contraintes entre elles par des relations qui toutes ont la forme générique $y \leq x + A^*_{x,y}$ ;
— on souhaite déterminer les valeurs prises par ces variables, de telle sorte que les fenêtres de temps associées aux variables de Y demeurent les plus grandes possibles.

Il comporte cependant une différence sensible. En effet, il nous manque des relations directes entre les variables de Y et celles de X : les quantités $INSER(i, \Gamma)$ sont définies sans référence explicite à des points d'ancrage. Afin de pallier cette difficulté, nous utilisons le fait que les quantités $INSER(i, \Gamma)$ sont directement associées, aux ensembles *NbCarLibre(i)* et aux listes $LibreXY(i)$. Nous associons donc, et il s'agit assurément d'une simplification, à chaque demande virtuelle $i$ (on se limite à celles pour lesquelles la quantité $INSER(i, \Gamma)$ est non nulle) le véhicule $k = k(i)$ et les points d'ancrage $x = x(i)$, $y = y(i)$ de $\Gamma_k$ qui fournissent la meilleure valeur $INSER2(i, \Gamma_\kappa, x, y)$. On peut alors créer, pour chaque $i \in DV$, les contraintes 6.5, 6.6, 6.7 et 6.8. (E2)

$$t(x) + Dist(x, o_i) \leq t(o_i) \tag{6.5}$$

$$t(o_i) \leq t(Succ(x, \Gamma_k)) + Dist(o_i, Succ(x, \Gamma_k)) \tag{6.6}$$

$$t(y) + Dist(y, d_i) \leq t(d_i) \tag{6.7}$$

$$t(d_i) \leq t(Succ(y, \Gamma_k)) + Dist(d_i, Succ(y, \Gamma_k)) \tag{6.8}$$

On se retrouve alors avec un ensemble $Z$ de couples de variables $t(o_i)$, $t(d_i)$, $i \in DV$, ces couples n'étant liés entre eux par aucune contrainte. On procède alors selon le schéma suivant en 2 étapes, qui nous permet de traiter, de façon heuristique, notre problème des rendez-vous comme une succession de problèmes Bi-Fenêtre :

— étape 1 : on travaille sur les quantités $INSER(i, \Gamma)$, $i$ dans $DV$, et on calcule des fenêtres $F^*(x) \subseteq F(x)$, $x \in X$, à l'intérieur desquelles les quantités $INSER(i, \Gamma)$ sont constantes ;
— étape 2 : on minimise *Eval*, les quantités $t(x)$ prenant leurs valeurs dans les intervalles $F^*(x)$.

La deuxième étape se fera exactement de la même façon que dans le cas du problème DARP standard, tel qu'il a été vu dans les premiers chapitres. Il nous reste donc ici à expliquer comment procéder dans la première étape. Celle-ci s'effectue en fait en une boucle principale induite par le parcours de l'ensemble $DV$. A chaque étape, on applique le processus Bi-Fenêtre, à l'exception de la dernière instruction qui fixe les valeurs $t(x)$, à partir de l'état courant des fenêtres $F(x)$. L'application de ce processus restreint les fenêtres $F^*(x)$ (cf. algorithme 26).

**Algorithme 26** Rendez-Vous

1: créer les ensembles de variables $X$ et $Y$ ; créer les contraintes (E2) ;
2: **Pour tout** $x \in X$ **Faire**
3: $F^*(x) \leftarrow F(x)$ ;
4: **Fin pour**
5: **Pour tout** $i \in I$ **Faire**
6: Appliquer, en omettant la dernière instruction, le Processus Bi-Fenêtre aux variables de $X$, aux fenêtres $F^*(x), x \in X$, et aux deux variables $o_i$ et $d_i$ : l'application de ce processus rétrécit les fenêtres $F^*(x), x \in X$ ;
7: **Fin pour**
8: minimiser *Eval* sur les fenêtres $F^*(x)$ en utilisant les procédés décrits en chapitre 3 ;

**Théorème 3**

Le processus Rendez-Vous fonctionne de telle manière que, pour tout $i$ dans $DV$, la valeur maximale de $INSER2(i, \Gamma_k, x(i), y(i)), t(x) \in F^*(x)$, demeure constante.

**Remarque**

Dans le processus ci-dessus, il est important aussi de remarquer que le problème de fixation de dates de rendez-vous ne concerne ni les nœuds $DepotD_k$ ni les nœuds $DepotA_k$. On se contente, pour les variables qui correspondent à ces nœuds du réseau complet, de gérer leurs fenêtres de temps par propagation de contraintes. Le processus Bi-Fenêtre doit donc être adapté en conséquence.

### 6.2.5 DARP dynamique : Synthèse

Le cadre dynamique se caractérise par le fait que le flot de demandes arrive au fur et à mesure et que les décisions de routage doivent être prises en conséquence. Ne pouvant avoir la certitude que toutes les demandes puissent être satisfaites, nous considérons ici qu'une demande peut être exclue.

Formellement, l'algorithme de gestion dynamique qui pilote le système, considère, à un instant donné $t$, un état de l'ensemble des tournées $\Gamma$ du système défini par les routes en cours, un état $DR$ des demandes non traitées, une représentation probabiliste $DV^*$ des demandes à venir, et décide, soit de ne rien faire, soit de traiter certaines demandes de $DR$ et rejeter les autres. Le traitement des demandes acceptées implique l'affectation de ces demandes aux routes $\Gamma_k$, ainsi que la détermination des dates de rendez-vous. Dans la pratique, il est souhaitable de scinder la spécification d'un tel algorithme en deux :

— une partie DEC qui déclenche ou non le processus de traitement des demandes en cours ;
— une partie ROUTE qui gère les demandes en cours.

L'enchaînement de ces 2 processus est repris par le diagramme d'états-transitions de la figure 6.6. Il contient en particulier un sous-processus MAIN-ROUTE qui résout ce que nous avons appelé modèle DARP local - MAIN-ROUTE.

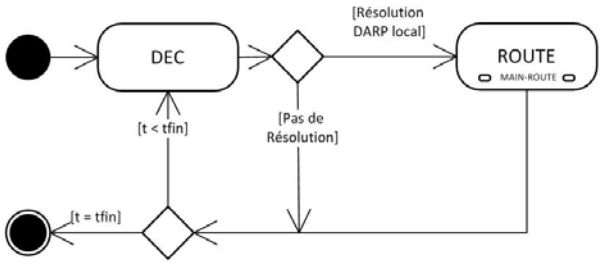


FIGURE 6.6 – Enchaînement des processus DEC et ROUTE

#### Le module de planification ROUTE

Le processus principal ROUTE considère :

— le réseau virtuel courant $X$ ;
— l'état des véhicules à l'instant $t$ ;
— un ensemble de demandes réelles $DR$, non pré-traitées, ce qui signifie que les $o_i$ et $d_i$ de chaque $i$ de $DR$ renvoient à un lieu géographique réel.

Il procède en 3 étapes :

— $1^{\text{ère}}$ étape : Prétraitement :
  — il met à jour le réseau virtuel courant $X$ en retirant les nœuds qui correspondent à des transactions déjà effectuées par les véhicules. Ceci a pour effet de faire disparaître certains nœuds de $X$ ;

— il convertit les demandes de $DR$ en nœuds $o_i$, $d_i$, $i \in DR$ du réseau virtuel $X$ : ceci revient à créer de nouveaux nœuds actifs de $X$ ;
— il remet à jour la demande virtuelle $DV^*$, ceci amenant à supprimer de $X$ les anciens nœuds associés à cette demande virtuelle et à en créer de nouveaux ;

— 2ème étape : le processus MAIN-ROUTE (cf. algorithme 27) peut s'appliquer grâce aux entrées alors disponibles. Il prolonge les tournées courantes, et rejette certaines demandes ;
— 3ème étape : les sorties de MAIN-ROUTE sont décodées de façon à pouvoir être comprises par les véhicules et les usagers.

**Algorithme 27** MAIN-ROUTE

**Pour** i de 1 à R **Faire**
  $continu \leftarrow VRAI$ ; $D1 \leftarrow DR$ ; *D-Rejet* $\leftarrow Nil$ ;
  **Tant que** *continu* **Faire**
    sélectionner $i_0$ dans $D1$ et tester son insertion pour tout $k$ dans $NbCarLibre(i_o)$ et tout $(x, y)$ dans $LibreXY(i_0)_k$ ; suivant le résultat des tests, pratiquer l'insertion $Inserted(\Gamma, i_0, k, x, y)$ qui permet de minimiser la quantité $Eval(Inserted(\Gamma, i_0, k, x, y), t) - \mu.\sum_{i \in D1\text{-}\{i_0\}} \Phi(INSER(i, Inserted(\Gamma, i_0, k, x, y))) - \alpha.\sum_i p_i.\Phi(INSER(i, Inserted(\Gamma, i_0, k, x, y)))$ ou bien placer $i_0$ dans *D-Rejet* ;
    **Si** $i_0$ a été insérée **Alors**
      retirer $i_0$ de $D1$ ; pour tout $i$ dans $D1$, mettre à jour les objets $NbCarLibre(i_o)$, $LibreXY(i_0)_k$, $INSER(i, \Gamma)$, $INSER1(i, \Gamma_k)$ et $INSER2(i, \Gamma_k, x, y)$ ;
    **Fin si**
    **Si** $D1 = Nil$ ou $NbCarLibre_i$ est vide pour tout $i$ de $D1$ **Alors**
      $continu \leftarrow FAUX$ ;
    **Fin si**
  **Fin tant que**
  retenter l'insertion des demandes de *D-Rejet* par ordre inverse de leur exclusion ;
  calculer les rendez-vous $t(o_i), i \in DR - D\text{-}Rejet$ de façon à minimiser $Eval(\Gamma, t) - \alpha.\sum_i p_i \Phi(INSER(i, \Gamma))$, en appliquant les procédures décrites en 6.2.4 ;
**Fin pour**

#### Le module déclencheur DEC

Le processus DEC a vocation à être appelé à intervalles réguliers très resserrés. Il doit donc prendre en entrée une quantité réduite de données de mise à jour rapide, de façon à pouvoir effectuer un traitement qui sera essentiellement à base de règles. Les éléments d'Input qui vont déterminer le résultat de DEC seront principalement le nombre de demandes en attente et l'urgence de ces demandes, c'est-à-dire, pour chaque demande $i$, l'écart entre l'instant courant et le milieu de la fenêtre associée à $o_i$. Les critères de qualité auxquels le résultat de DEC est soumis est en soi difficile à formaliser : c'est seulement dans le cadre d'une simulation mettant en jeu la combinaison (DEC, ROUTE) qu'il sera possible de se rendre compte du niveau de qualité effective des résultats produits par DEC.

Toutefois, on peut constater un certain nombre d'éléments :

— la qualité des décisions de routage sera d'autant plus élevée que l'on a été en mesure de traiter certaines demandes de façon groupée ; un traitement "à l'aveugle", demande par demande, n'est donc pas souhaitable ;
— la façon dont on abordera le problème de façon pratique dépendra toutefois beaucoup du contexte et notamment des procédures mises en place pour le dialogue entre le demandeur et le système. Ces procédures fixeront les délais dans lesquels les demandeurs ont à recevoir des réponses à leurs demandes, et les écarts de temps admissibles par le système entre la formulation d'une demande et la réalisation de celle-ci (date du rendez-vous).

On peut dès lors envisager pour DEC différentes stratégies simples :

**Stratégie 1 dite "Rigide"** : DEC est appelé à intervalles fixes d'amplitude $\eta$, et fournit toujours une réponse positive. Le seul point demeurant à préciser concerne alors la valeur de $\eta$. On voit que :
— $\eta$ doit être sensiblement plus grand que le temps $\delta$ nécessaire pour le fonctionnement de ROUTE, ceci incluant les pré-traitements et les post-traitements, ainsi que le temps nécessaire à la transmission des résultats de ROUTE aux véhicules et aux usagers ;
— $\eta$ doit être compatible avec les délais imposés aux usagers pour la satisfaction de leurs demandes (écart entre la valeur $\text{Min}(o_i)$ et la date d'émission de la demande $i$) ainsi qu'avec les délais dans lesquels le système est supposé fournir une réponse aux usagers. Bien sûr, le mode de communication envisagé pour ces réponses (Internet, Téléphone, Borne d'Appel/Réponse...) conditionne fortement la valeur de ces délais.

**Stratégie 2 dite "Adaptative"** : le processus DEC sera déclenché à tout petits intervalles de durées $\tau$ ($\tau$ sensiblement inférieur à $\delta$) de façon à fournir, suivant le cas, une réponse OUI ou NON, reflétant l'état des données suivantes :
— valeur courante de l'instant $T$ ;
— valeur $T_R$ de l'instant auquel a eu lieu le dernier appel de ROUTE ;
— nombre $N$ de demandes en attente et volume $L$ de la charge en attente ;
— urgence minimale $U$ des demandes en attente : l'urgence d'une demande $i$ sera la valeur $\text{Max}(o_i)$.
On pourra par exemple déclencher DEC si l'expression booléenne suivante, avec $Seuil$, $Seuil1$, $Seuil2$ étant 3 paramètres et $H$ une fonction croissante et positive, est vraie :

$$(H(L,N) \geq Seuil) \vee ((U-T) \leq Seuil1) \vee (T - T_R \leq Seuil2) \tag{6.9}$$

**Stratégie 3 dite "Flexible"** : on différenciera les demandes selon leur niveau d'urgence (et éventuellement selon d'autres critères tels que la nature de leur relation contractuelle au système), et on appliquera à chaque groupe ainsi identifié un traitement spécifique. On peut par exemple, envisager une différenciation des demandes entre demandes urgentes ($U - T \leq Seuil1$) et demandes standards, et envisager, sur le mode de l'approche "Rigide", une valeur $\Delta$-*Urgent* marquant le délai entre 2 traitements consécutifs des demandes urgentes et une valeur $\Delta$-*Standard* marquant le délai entre 2 traitements consécutifs des demandes standards. On fera en sorte que $\Delta$-*Standard* soit un multiple entier de $\Delta$-*Urgent* et que $D\text{-}Urgent$ soit sensiblement plus grand que $\delta$. En fait, il pourra alors être opportun d'avoir 2 versions de ROUTE, l'une adaptée aux demandes standards (temps d'exécution $\delta$-*Standard*), et l'autre accélérée, simplifiée, et adaptée aux demandes urgentes (temps d'exécution $\delta$-*Urgent*). On devra alors avoir $\Delta$-*Standard*, $\delta$-*Standard*, $\Delta$-*Urgent*, $\delta$-*Urgent* et $\Delta = r.\delta$, où $r$ est un entier.

### 6.2.6 Simulation

L'évaluation des procédures ROUTE et DEC ne peut se faire efficacement que par le biais de simulations. Le but de la présente section est donc de proposer un cadre pour celles-ci.

On fait ici abstraction des aspects directement liés à la communication entre systèmes et véhicules, et, surtout, entre systèmes et usagers. Ces points, qui renvoient notamment à la conception de protocoles de communication système/usagers à même de rendre l'ensemble du dispositif DARP robuste face aux différentes défaillances (retards des véhicules ou des usagers, abandon de requête de la part de l'usager...), constituent en eux-mêmes un problème difficile, requérant une étude spécifique et explicitement liée au contexte (technologies de communication mise en œuvre, type de véhicules, nature de l'espace géographique concerné...). On ne s'occupe pas non plus du comportement interne des véhicules. On limitera donc l'introduction des aléas à deux éléments :

— la génération des demandes, considérées dès lors comme totalement fiables une fois qu'elles ont été générées ;
— les temps de parcours des véhicules, qui pourront parfois avoir de l'avance ou du retard, sans considérer toutefois l'hypothèse de pannes.

On se place donc dans la perspective d'une simulation stochastique à événements discrets (cf. la gestion du temps du chapitre 3), et, pour des raisons de commodité, à pas de temps constant. Ceci n'est clairement pas un choix optimal, car, dans le cas d'un pas de temps réduit, on risque de gérer beaucoup d'instants où il ne se passera presque rien.

#### 6.2.6.1 Etats du système

A un instant donné t, considéré à l'intérieur d'un horizon de simulation $[0, \Delta_{Max}]$, l'état E = E(t) du système sera défini par :

— la position des véhicules $k = 1..K$ dans un réseau réel G = (V, A) ; les nœuds correspondent aux dépôts et aux possibles $o_i$, $d_i$. Les arcs désignent les routes élémentaires que peuvent suivre les véhicules pour se déplacer. Ce réseau G constitue donc une approximation d'un réseau urbain ou périurbain réel. Chaque arc e dans A possède une longueur h(e). La position géographique réelle d'un véhicule est donc identifiée soit par un nœud de G soit par un arc e de G et un coefficient entre 0 et h(e) indiquant la part de l'arc e qui a été parcourue par le véhicule. Le réseau G n'est pas forcément symétrique ;
— le plan de route courant $\Gamma_k$ de chaque véhicule $k = 1..K$, c'est-à-dire la donnée de la charge courante du véhicule et de la liste de ses prochains points de passage (avec, pour chacun de ceux-ci, la date prévue pour le passage et la description de l'action que le véhicule doit effectuer : charger une ou des demandes, décharger une ou des demandes...) ;
— les demandes $DR$ couramment en stock (en attente de traitement par DEC et ROUTE) ;
— le graphe complet auxiliaire $X$ (réseau virtuel), utilisé par la procédure ROUTE : les nœuds de $X$ sont exactement les nœuds $DepotA_k$, $k = 1..K$, les nœuds origines et destinations des demandes virtuelles de $DV$ et des demandes réelles de $DR$, et les nœuds $o_i$ et $d_i$ correspondant à des demandes déjà traitées, mais encore présents dans le plan de route d'un véhicule ;
— l'état courant Statut-ROUTE du processus ROUTE : Actif (1) ou Inactif (0), et, dans le cas où ROUTE est actif, l'instant de déclenchement de ce processus ;
— la date $t$-Route du dernier déclenchement de la procédure ROUTE.

#### 6.2.6.2 Transitions

Le passage d'un instant $t$ à l'instant $t+\tau$, où $\tau$ est le pas de temps, est susceptible d'induire le déclenchement de plusieurs transitions qui agiront sur l'état E($t$) et fourniront l'état E($t + \tau$). Ces transitions sont de 3 types :
— logiques et physiques,
— décisionnels et de contrôle,
— par évènements aléatoires.

##### Transitions logiques et physiques

Ces transitions concernent la position des véhicules et la représentation du réseau virtuel.

**Position des véhicules.** Un véhicule situé sur un arc $e$, à une distance $h$ de son origine, avance à une distance $h' = h+u$ : $u$ est tiré aléatoirement. La quantité $h+u$ ne doit pas excéder la longueur $h(e)$. Un véhicule situé en un nœud $v$, dont le plan de route implique de se déplacer vers un nœud voisin $v'$, et qui n'a pas d'opération de chargement ou de déchargement à réaliser en $v$, se déplacer de la même façon.

Un véhicule situé en un nœud $v$, dont le plan de route implique de charger ou de décharger en $v$ puis de se déplacer vers un nœud voisin $v'$, se déplacer de la même façon à une distance $u'$ sur l'arc $[v, v']$, $u'$ tiré au sort avec une loi prenant en compte le service en $v$.

Si le véhicule doit charger en $v$, et est en avance (au-delà de la marge d'avance autorisée) sur le temps de rendez-vous prévu en $v$, alors il attend (déplacement nul). Ce temps fait référence aux *Waiting times* introduits dans le chapitre 3. Si le véhicule doit charger en $v$, et qu'il est en retard (au-delà de la marge de retard autorisée), alors la demande est retirée du plan de route ainsi que du réseau virtuel et est comptabilisée en *Erreur-Retard*.

Les distances DIST$(x, y)$, le pas de temps $\tau$ et les quantités $u$ et $u'$, sont bien sûr définies de façon cohérente.

**Réseau virtuel $X$.** Chaque fois que la transition sur les plans de route commande la suppression de nœuds dans les plans de route, ces nœuds sont également retirés du réseau virtuel $X$. De même, lorsque la transition sur les demandes commande la création de nouvelles demandes, celles-ci sont incorporées dans le réseau virtuel. Dans le cas où Statut-Route = 1, les nœuds qui devraient être retirés de $X$ du fait de la mise à jour des plans de route, sont stockés dans une mémoire provisoire Buffer-nœud.

#### Transitions par évènements aléatoires

Ces transitions concernent la génération aléatoire des demandes. Elles sont commandées par une représentation probabiliste des demandes également utilisée pour le calcul des demandes virtuelles. A chaque instant $t$, de nouvelles demandes sont susceptibles d'être émises conformément à cette représentation.

#### Transitions décisionnelles et de contrôle

**Plans de route.** La transition sur le plan de route se contente d'agir sur les plans de route de façon mécanique en répercutant les opérations de charges et décharges réalisées par les véhicules : dans l'hypothèse donc où un véhicule $k$ est situé à l'instant $t$ en un nœud $v$, et si son plan de route implique de charger ou décharger en $v$ puis de se déplacer vers un nœud voisin $v'$, les nœuds $o_i$ et $d_i$ concernés sont retirés de son plan de route et sa charge courante est mise à jour.

**Transitions relatives à la procédure ROUTE.** A chaque instant $t$ tel que Statut-Route = 0 (procédure ROUTE inactive), la procédure DEC est déclenchée, préalablement à toute autre transition, et considérée comme instantanément exécutée. Dans le cas où le résultat de DEC = 0, ROUTE demeure inactive.

Si Statut-Route = 0 et DEC = 1 alors, préalablement à la mise œuvre de toute autre transition :

— Statut-Route est mis à 1 ;
— la date t-Route de dernier lancement de ROUTE est mise à $t$ ;
— la demande aléatoire $DV$ nécessaire pour l'exécution de MAIN-ROUTE (localisée à l'intervalle $[t,\ t\ +\ \Delta]$) est générée ainsi que l'ensemble $DV$ actif correspondant ;
— dans le réseau virtuel X, les nœuds relatifs à $DV$ sont activés ;
— MAIN-ROUTE est exécutée, en recevant comme input le réseau virtuel X courant, la demande aléatoire restreinte $DV$, l'ensemble des demandes $DR$ couramment en stock, et l'ensemble des routes courantes $\Gamma$, représentées en référence à X ;
— $DR$ est vidé. La transition sur les demandes est alors activée. La transition sur le réseau virtuel est gelée.

Si Statut-Route = 1, alors, dépendant de $t$ et de $t$-Route, Statut-Route passe à 1 ou 0 de façon aléatoire (ou déterministe si l'on estime que la durée d'exécution de ROUTE est fixe et égale à un nombre $\delta$ multiple entier de $\tau$), et ceci préalablement à toute autre transition, hors la transition relative à DEC. Il ressort que DEC, qui ne peut être lancé quand Statut-Route = 1, ne peut non plus l'être à l'instant $t$ quand Statut-Route repasse à 0. Si Statut-Route reste à 1, alors, les transitions relatives au réseau virtuel $X$ demeurent gelées.

Quand, à l'issue du processus précédent, Statut-Route repasse à 0, alors :
— le réseau virtuel X est vidé des demandes qui ont été rejetées ;
— il est enrichi des demandes de $DR$ ;
— il est vidé des nœuds de Buffer-nœud ;
— les plans de routes construits par ROUTE sont reformatés en format géographique, et constituent les nouveaux plans de route des différents véhicules. Dans le cas où, pour une route donnée $\Gamma_k$, il existe dans $\Gamma_k$ une demande nouvelle $i$ pour laquelle le passage sur le nœud $o_i$ a été planifié avant un nœud de Buffer-nœud qui n'est pas de la forme $d_i$, où $i$ est associée à une erreur-retard, alors une erreur-planning est comptabilisée et la demande $i$ est retirée de $\Gamma_k$ et considérée comme rejetée. Cette situation traduit en effet une inconsistance entre les temps d'exécution de ROUTE et le rythme auquel avancent les véhicules.
— Buffer-nœud est vidé. Dans le cas où, pour une route donnée $\Gamma_k$, il existe dans $\Gamma_k$ une demande nouvelle $i$ pour laquelle le passage sur le nœud $o_i$ a été planifié avant un de ces nœuds de Buffer-nœud, alors une erreur-planning est comptabilisée et la demande $i$ est retirée de $\Gamma_k$ et considérée comme rejetée.

**Remarque : A propos des Aléas**

La gestion des aléas se fait par tirage au sort indépendant (simplification importante) et porte exclusivement sur :
— la génération, à chaque instant des nouvelles demandes pour $DR$ ;

— la détermination du temps de calcul effectif $\delta$ pour ROUTE, qui doit être cohérent avec le pas de temps $\tau$ et avec les mesures de temps de parcours des véhicules ;
— la détermination, à chaque instant $t$, des amplitudes de déplacement réelles des véhicules.

Demandes virtuelles et demandes réelles ont bien sûr à être liées. La demande virtuelle générée à un instant $t$ fait l'objet d'un processus de génération aléatoire, et diffère donc à priori de celle qui a pu être utilisée lors des appels antérieurs de ROUTE. S'il n'en était pas ainsi, cette demande virtuelle viendrait à jouer le rôle d'une demande réelle. Elle est donc dès lors générée sur un mode similaire aux demandes réelles, les deux exprimant le même phénomène aléatoire. Pour autant, l'une n'est pas une simple anticipation de l'autre, car sinon cela reviendrait à supposer que l'on est capable de prévoir les demandes de façon exacte.

#### Les indicateurs de performance du système simulé

On s'intéresse en premier lieu aux indicateurs reflétant directement des critères de qualité imposés au niveau statique, c'est-à-dire :

— les critères contenus dans *Eval* ;
— le nombre de demandes rejetées par le processus ROUTE ;
— la valeur *Erreur-Retard* ou nombre de demandes non satisfaites pour cause de retard des véhicules ;
— la valeur *Erreur-nœud* ou nombre de demandes non satisfaites du fait d'inconsistance de la planification.

### 6.2.7 Résultats expérimentaux

#### Résumé

Une première série de tests cherche à évaluer les difficultés induites par le contexte dynamique. Nous comparons les résultats (Indicateurs) obtenus en contexte effectivement dynamique, avec ceux que nous aurions pu obtenir en supposant toutes les demandes connues dès l'instant 0.

Une seconde série d'expérimentations vise à évaluer la robustesse de notre système s'il comprend, au niveau de la résolution du DARP local, les techniques développées dans ce chapitre basées sur la mesure d'*Insérabilité*. Nous comparons en effet le nombre de demandes ayant pu être insérées avec ou sans cette notion de robustesse.

#### Environnement et protocole expérimental

Comme pour les autres chapitres, les différentes méthodes sont développées en C++ et exécutées sur machines 64bits. Pour ces tests, le nombre de réplications de la procédure d'insertions successives est systématiquement de 50.

#### Les instances

Bien que l'ensemble des tests soit réalisé au travers d'une simulation, les demandes sont ici générées en amont. C'est en effet la date d'émission générée qui va déterminer son arrivée dans le système. Cela signifie que les instances sont générées sur le mode des instances statiques, à la différence que chaque demande est accompagnée d'une date d'émission.

**Série 1.** Cette série contient 3 instances de base, chacune donnant lieu à 8 expérimentations :

- suivant la façon plus ou moins stricte dont sont gérés les rendez-vous : pour une instance de base T, le suffixe RDV ou Large indique respectivement que les rendez-vous permettent un retard de 5 minutes ou bien que les fenêtres restent telles qu'elles ont été réduites par le processus de propagation de contraintes ;
- suivant le rythme des appels locaux à ROUTE et, de façon induite, le degré de "dynamicité" imposé au système, chaque instance de base T donne naissance aux versions $T_j, j = 1000, 333, 50, 15$ : $j$ donne l'amplitude de la période entre 2 appels consécutifs de ROUTE, et indique que ROUTE a une vision exacte sur toutes les demandes émises sur un intervalle d'amplitude $j$ (si $j = 1000$, on est ramené au cas statique). En fait, on peut aussi comprendre la valeur $j$, en considérant que la date d'émission d'une demande pour $T_j$ est celle correspondant à $T$, diminuée de $j$.

Pour ces trois instances, à l'instar des instances des chapitres précédents traitant du DARP, les demandes couvrent l'espace d'un carré de côté 20. Alors que la capacité des véhicules est de 6 (à l'image de ceux du type VIPA), les demandes ont une charge aléatoire entre 1 et 3.

Les deux premières séries comprennent 100 et 200 demandes à satisfaire pour respectivement 4 et 5 véhicules, la troisième considère 300 demandes pour 5 véhicules. Les durées maximum des tournées et des demandes sont générées pour donner des contraintes lâches. Les deux fenêtres de temps sont par contre plus serrées : 30 minutes pour chacune des origines et destinations des deux premières instances puis des amplitudes de 15 et 60 minutes partagées uniformément dans chaque couple origine/destination de la troisième instance.

**Série 2.** La génération de cette deuxième série d'instances reprend les grandes lignes de la première, c'est-à-dire : un système évoluant sur 1000 minutes, un carré de côté 20, une flotte de véhicules de capacité 6 et des chargement de 1, 2 ou 3. Les temps de service sont à nouveau de 1 et ceci pour chaque type de nœud (hors dépôts) et les dates d'émissions sont toujours entre 0 et 300 minutes avant la borne MIN de la fenêtre de temps de l'origine.

Nous considérons alors la distribution des demandes virtuelles comme "juste" (c'est à dire que les demandes virtuelles en temps t apparaîtront plus tard, telles quelles, comme demandes réelles). Cependant, l'ensemble de ces futures demandes ne sont pas connues au moment de la résolution dite "locale". Le nom d'une instance de base a pour préfixe T selon la sémantique de la série 1. Le suffixe I ou B différencie les modes de résolutions, suivant que l'on a fonctionné "à l'aveugle" (B) ou en tenant compte de la demande virtuelle, de la mesure d'*Insérabilité* et des rendez-vous (I).

#### Résultats et analyses

**Série 1.** Les résultats obtenus sur les deux premières instances sont relevés en tableau 6.4 et 6.5, ceux pour la troisième instance le sont en tableau 6.6.

| Inst. | $T_{Insert}$ | $T_{Global}$ | $T_{Ride}$ | $Dist$ |
|---|---|---|---|---|
| T1000RDV | 100 | 2504 | 1847 | 1601 |
| T333RDV | 100 | 2697 | 1806 | 1703 |
| T50RDV | 98 | 2881 | 1878 | 1799 |
| T15RDV | 81 | 2676 | 1615 | 1552 |
| T1000Large | 100 | 2504 | 1847 | 1601 |
| T333Large | 100 | 2618 | 1709 | 1740 |
| T50Large | 100 | 2907 | 1794 | 1810 |
| T15Large | 88 | 2607 | 1413 | 1630 |

TABLE 6.4 – Performances des tournées pour le DARP dynamique - 100 demandes et 4 véhicules

| Inst. | $T_{Insert}$ | $T_{Global}$ | $T_{Ride}$ | $Dist$ |
|---|---|---|---|---|
| T1000RDV | 99.5 | 3565 | 3619 | 2891 |
| T333RDV | 96.0 | 3522 | 3417 | 2854 |
| T50RDV | 67.5 | 3044 | 2670 | 2337 |
| T15RDV | 62.0 | 2900 | 2397 | 2149 |
| T1000Large | 99.5 | 3565 | 3619 | 2891 |
| T333Large | 98.5 | 3572 | 3503 | 2967 |
| T50Large | 71.0 | 2524 | 2536 | 2172 |
| T15Large | 62.0 | 2571 | 2074 | 2131 |

TABLE 6.5 – Performances des tournées pour le DARP dynamique - 200 demandes et 5 véhicules

| Inst. | $T_{Insert}$ | $T_{Global}$ | $T_{Ride}$ | $Dist$ |
|---|---|---|---|---|
| T1000RDV | 85.3 | 3804 | 5191 | 3500 |
| T333RDV | 83.3 | 3712 | 5114 | 3380 |
| T50RDV | 40.3 | 2266 | 2009 | 1865 |
| T15RDV | 38.7 | 2456 | 2116 | 1956 |
| T1000Large | 85.3 | 3804 | 5191 | 3500 |
| T333Large | 84.0 | 3773 | 4924 | 3435 |
| T50Large | 51.3 | 2660 | 2657 | 2292 |
| T15Large | 42.0 | 2518 | 1979 | 2016 |

TABLE 6.6 – Performances des tournées pour le DARP dynamique - 300 demandes et 5 véhicules

Pour chacune des instances, nous pouvons remarquer l'effet qu'a l'intervalle de résolution du DARP local sur les résultats et notamment sur le taux de demandes insérées. Par exemple, lors des résolutions fixées toutes les 15 minutes des expériences liées au tableau 6.6, les différentes flottes de véhicules vont n'accueillir que la moitié des usagers, si l'on compare au cas T1000.

L'effet sur la réduction des fenêtres dues aux rendez-vous semble ici moins spectaculaire que ce à quoi l'on pouvait s'attendre en ce qui concerne le taux de demandes insérées. Remarquons toutefois que les résolutions 'Large' (i.e. sans processus de rétrécissement des fenêtres dû à l'émission des dates de service) n'ont jamais moins de demandes insérées que dans le cas des résolutions 'RDV'.

Dans le cas 'RDV', le processus se tourne vers un choix plus restreint de demandes, au risque d'augmenter significativement les différents coûts engendrés par les tournées. Par exemple, prenons la dernière résolution relevée dans le tableau 6.5, où l'on obtient le même taux d'insertion pour T15Large et T15RDV. Ici les $T_{Global}$ de T15RDV "explosent" : les tournées ont duré 300 minutes de plus qu'avec T15Large.

**Série 2.** Le tableau 6.7 indique les taux d'insertion obtenus, pour une même matrice de distance, selon différentes possibilités du couple $|D|$,$K$ (en ligne), et suivant les différentes stratégies I et B (en colonne).

| D | K | T50B | T50I | T333B | T333I | T1000B | T1000I |
|---|---|---|---|---|---|---|---|
| 200 | 2 | 38.0 | 49.5 | 43.0 | 50.5 | 46.0 | 51.0 |
| 300 | 3 | 25.0 | 38.7 | 48.3 | 52.6 | 51.3 | 53.7 |
| 250 | 4 | 45.6 | 57.6 | 66.4 | 73.2 | 69.2 | 74.4 |
| 350 | 5 | 39.7 | 47.7 | 62.6 | 68.3 | 67.1 | 69.4 |

TABLE 6.7 – $T_{Insert}$ selon les résolutions intégrant (I) ou pas (B) les procédés liés à l'*Insérabilité*

Nous remarquons que le GAP entre les deux taux des deux résolutions dépasse assez régulièrement les 10% dès lors que la procédure ROUTE est appelée toutes les 50 minutes. Cette différence s'amenuise dès lors que le nombre d'appels de ROUTE est moins important. Aussi, pour ce qui concerne le cas T1000, avec donc une seule résolution, subsiste-t-il des différences entre les deux techniques alors qu'aucune demande virtuelle n'a été générée. Ceci s'explique par les procédés d'anticipation des demandes restant à insérer par les méthodes développées en début de chapitre.

## Conclusion et perspectives

Les traitements du DARP par insertions successives des demandes, tels qu'ils ont pu être réalisés dans la littérature, souffrent d'un défaut : lorsqu'une demande est insérée dans un ensemble de tournées courantes, il est peu tenu compte de l'impact de cette insertion sur la difficulté qu'il pourra y avoir à traiter les demandes restant à insérer. Le but de cette recherche était donc d'aborder ce problème, et de montrer de quelle manière il est possible de prendre en compte les insertions à venir. Nous avons mis en place un calcul de l'*Insérabilité* des demandes puis son implication aux moments de la sélection d'une demande, de son exclusion potentielle, du choix de son emplacement dans une tournée et enfin du calcul des rendez-vous. Cette mesure permet l'anticipation des demandes restant à insérer dans les contextes statiques et dynamiques. Dans ce dernier cas, la mesure est également utilisée sur une représentation des demandes qui feront l'objet d'insertion lors du futurs appels de la procédure générale de résolution. C'est sur ce point qu'apparaît la notion de robustesse : les décisions en $t$ sont prises en partie selon les futures possibilités d'insertion.

# Chapitre 7

# DARP statique avec contraintes de fiabilité - DARPRel

## Introduction : le problème

Les progrès réalisés dans l'automatisation des véhicules nous amènent à considérer de nouveaux problèmes de tournées, qui intègrent de nouvelles contraintes. Le présent chapitre traite de l'intégration de la notion de fiabilité dans la supervision d'une flotte de véhicules autonomes proposant un service de transport à la demande (TaD) à petite échelle. Cette recherche de fiabilité est induite par les difficultés qu'ont des véhicules autonomes à réaliser les opérations liées aux chargements et déchargements : changement de route impliquant décélération/accélération, arrêt du véhicule, ouverture et fermeture des portes. Plus spécifiquement, nous nous intéressons aux véhicules VIPA évoluant, toujours dans le même sens, le long d'un circuit ou d'un axe horizontal. Un VIPA est un véhicule (commercialisé par LIGIER S.A.) qui se déplace sans conducteur, et sans l'aide d'un rail ou d'un fil de guidage, à l'aide d'une combinaison de techniques de vision artificielle et de géolocalisation. La figure 7.1 représente un circuit avec les trois catégories de routes utilisées par le VIPA : la voie principale, l'aire d'échanges de passagers et le point de stationnement. La figure 7.2 schématise le rapport entre ces trois types de voies et la vitesse du véhicule. Pratiquement, le circuit en question correspond à une zone réservée à l'intérieur d'un centre ville ou d'un espace industriel, et sa taille n'excède pas quelques kilomètres.

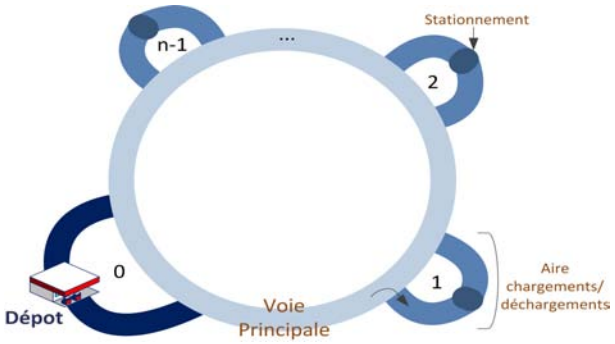


FIGURE 7.1 – Aires d'échanges, voie principale et dépôt d'une flotte de VIPA

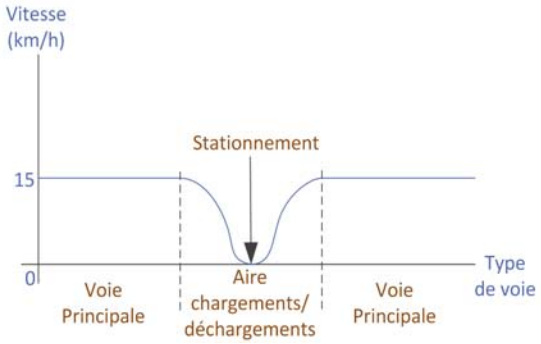


FIGURE 7.2 – Profil de la vitesse d'un VIPA

Les véhicules VIPA évoluant sur circuit fermé sont sujets à des demandes d'acheminement dont l'expression est similaire à ce qui prévalait dans les chapitres précédant : une demande $i \in D$ porte donc sur l'acheminement d'une charge $Q_i$ depuis un point $o_i$ (origine) de notre circuit vers un point $d_i$ (destination). Un véhicule chargeant la demande $i$ en $o_i$ doit obligatoirement la décharger dès qu'il passe en $d_i$ (pas de tour "pour rien"). Lorsqu'un véhicule procède à une opération de chargement ou de déchargement en un nœud $x$ du circuit, il quitte la piste principale pour s'engager sur l'aire de chargement/déchargement située en $x$. Quand il revient sur la piste principale, il n'est pas prioritaire par rapport aux véhicules circulant sur la piste principale. Ceci signifie qu'alors que les véhicules ne peuvent se doubler lorsqu'ils sont sur la piste principale, les opérations de chargement/déchargement peuvent modifier l'ordre dans lequel les véhicules évoluent sur le circuit. Du fait de l'absence de conducteur, les opérations de chargement/déchargement constituent la partie la plus vulnérable du système, à la fois au niveau du temps requis pour satisfaire les demandes, et, surtout, pour ce qui concerne les risques d'erreurs et de défaillance du système. Ceci nous amène à imposer, comme critère de qualité dominant pour une stratégie de gestion des demandes, que soit minimisé le nombre de fois où les véhicules ont à procéder à des opérations de chargement/déchargement. On dira alors que l'on cherche à minimiser le "nombre des arrêts". Si l'on se réfère aux temps de parcours, tant des véhicules que des usagers (les quantités $T_{Global}$ et $T_{Ride}$), on voit que ces temps, du fait de la vitesse constante des véhicules sur la voie principale et de l'interdiction des "tours pour rien", sont corrélés au nombre des arrêts. De fait, minimiser leur nombre permettra aussi de minimiser les temps de parcours des véhicules et des usagers.

Le problème ainsi posé constitue donc une version spécifique du DARP que nous nommons **DARPRel**. Pratiquement, ce DARP avec "minimisation des arrêts", a vocation à être traité, compte tenu de l'amplitude réduite des déplacements en jeu, dans un contexte dynamique, voire relevant du Temps Réel. Nous allons cependant, dans un premier temps, étudier ce problème dans la perspective de la recherche, via un modèle linéaire en nombres entiers, de solutions exactes, et afin de disposer de *benchmarks* pour l'évaluation d'algorithmes approchés. Nous étudions ici ce problème en considérant d'abord une version simplifiée, dite "en ligne" (les véhicules n'effectuent qu'un seul tour de circuit), avant d'aborder le cas général. Nous nous

limitons également dans un premier temps au cas où les demandes sont exprimées sous la forme de tuples $(o_i, d_i, Q_i)$ avant d'envisager la prise en compte de fenêtres de temps et de bornes sur les durées d'acheminement.

Que ce soit dans l'établissement de tournées de véhicules, mais aussi en ce qui concerne la production, le problème de la minimisation des arrêts, tel qu'il est décrit dans ce chapitre, est absent de la littérature scientifique. Les problèmes de plannings d'ascenseurs ont une certaine proximité avec le nôtre : nous appelons d'ailleurs ce système de véhicules autonomes un "ascenseur horizontal". Les problèmes d'ascenseurs verticaux ont en revanche donné lieu à plusieurs travaux. Nous pouvons mentionner les algorithmes génétiques ([Fujino et al. (1997)] et [Kim and Moon (2001)]), les algorithmes tirés des réseaux de neurones artificiels ([So et al. (1995)] et [Dewen et al. (1997)]) ou encore des méthodes basées sur les systèmes intelligents flous ([Igarashi et al. (1994)]). Certains travaux mettent l'accent sur les aspects dynamiques comme [Kwon et al. (2014)] qui, à l'instar du chapitre précédent sur le DARP dynamique, cherche à développer un système robuste. Ils cherchent à minimiser tout d'abord le temps d'attente moyen et maximum puis la consommation énergétique. [Tanaka et al. (2005)] ont résolu le problème dynamique mono-voiture. Ils ont mis au point un algorithme de *branch-and-bound* couplé avec de la relaxation lagrangienne à l'image d'une des techniques présentées dans les sections suivantes. Les systèmes de contrôle des *Automated Guided Vehicles* (AGV) sont peut-être ceux qui se rapprochent le plus de notre problème. On peut se reporter à la bibliographie [Le-Anh and De Koster (2006)] qui traite notamment de l'ordonnancement des AGV mais aussi du problème du nombre de véhicules nécessaires à la satisfaction de l'ensemble des tâches. [Meer (2000)] explique que le nombre de véhicules déployés influence fortement la performance des systèmes automatisés. Leur prix est pris en compte mais il est tout aussi vrai, et c'est le cas des VIPA, que moins il y a de véhicules, plus le système est fiable. [Bilge and Tanchoco (1997)] a, quant à lui, montré au travers d'une simulation que les *multi-load capacity vehicles* (caractéristique que l'on peut associer aux VIPA du fait qu'ils peuvent transporter plusieurs personnes de requêtes différentes) sont plus efficaces que les *unit-load vehicles*. [Meer and Koster (1999)] surenchérit en montrant que les *multi-load capacity vehicles* manifestent une plus grande efficacité par rapport aux *unit-load vehicles* quand le point de chargement est identique. La construction de routes au niveau des AGV est similaire au *Pick up and Delevery Problems with Time Windows* ([Le-Anh and De Koster (2006)]), le chapitre 2 en fait l'état de l'art.

## 7.1 Modèles mathématiques statiques

Cette première section regroupe les différents modèles mathématiques du DARPRel. Ils sont partagés en deux ensembles selon que les véhicules évoluent sur un axe horizontal ou sur circuit.

### 7.1.1 Modèles sur topologie Ligne

Au début, nous ne tenons pas compte des contraintes de temps que nous trouvons dans les problèmes classiques de *Dial-a-Ride*. On considère donc un axe horizontal où sont alignés $S$ stations/arrêts, un ensemble $D$ de demandes $i$, chacune composée d'un 3-uplet $o_i, d_i, Q_i$ où $o_i$ et $d_i$ sont respectivement l'origine et la destination puis $Q_i$ la quantité associée à transporter, et $K$ véhicules homogènes de capacité $CAP$. Le but du problème est alors de distribuer les demandes de $D$ sur les $K$ véhicules de façon à minimiser le nombre d'arrêts en station tout en respectant la capacité des véhicules. Nous dérivons de $D$ : 2 vecteurs $a = (a_{s,i}, s = 0..S, i \in D)$ et $b = (b_{s,i}, s = 0..S, i \in D)$ à valeurs dans [0,1] déterminées comme suit :

- $a_{s,i} = 1$ ssi la station $s$ est l'origine de la demande $i$,
- $b_{s,i} = 1$ ssi la station $s$ est la destination de la demande $i$.

Nous obtenons alors un premier modèle $P_{MinArretL1}$ pour notre problème DARPRel à l'aide de 3 vecteurs inconnus :

- $Z = (Z_{i,k}, i \in D, k = 1..K)$, à valeurs dans [0,1] : $Z_{i,k} = 1$ ssi la demande $i$ est servie par le véhicule $k$ ;
- $T = (T_{k,s}, k = 1..K, s = 0..S)$, à valeurs dans [0,1] : $T_{k,s} = 1$ ssi le véhicule $k$ s'arrête en $s$,
- $ChT = (ChT_{s,k}, s = 0..S, k = 1..K)$, à valeurs entières : $ChT_{s,k}$ représente la charge du véhicule $k$ quand il vient de passer la station $s$.

$Z$, $T$ et $ChT$ doivent alors minimiser la quantité "nombre d'arrêts" :

$$\sum_{k=1..K} \sum_{s=0..S} T_{k,s} \tag{7.1}$$

tout en satisfaisant les contraintes :

$$\sum_{k \in (1..K)} Z_{i,k} = 1, \forall i \in D \tag{7.2}$$

$$a_{s,i} + Z_{i,k} - 1 \leq T_{k,s}, \forall s \in (0..S), i \in D, k \in (1..K) \tag{7.3}$$

$$b_{s,i} + Z_{i,k} - 1 \leq T_{k,s}, \forall s \in (0..S), i \in D, k \in (1..K) \tag{7.4}$$

$$ChT_{s,k} = ChT_{s-1,k} + \sum_{i \in D} a_{s,i}.Q_i.Z_{i,k} - \sum_{i \in D} b_{s,i}.Q_i.Z_{i,k}, \forall s \in (0..S), k \in (1..K) \tag{7.5}$$

$$ChT_{s,k} \leq CAP, \forall s \in (0..S), k \in (1..K) \tag{7.6}$$

La première contrainte indique que la demande doit être satisfaite une fois et une seule. Les deux suivantes expriment que le véhicule prenant en charge une demande $i$ s'arrête nécessairement, et une seule fois, aux deux nœuds *origine* et *destination* de $i$. La quatrième exprime l'équilibre de la charge avec les volumes $ChT_{s-1,k}$ et $ChT_{s,k}$ transportés par les véhicules provenant du nœud précédent $s-1$ et du nœud courant $s$, soumis aux variations résultant des opérations de chargement et de déchargement sur ce dernier nœud. La dernière contrainte implique le respect de la capacité des véhicules.

Nous pouvons proposer un deuxième modèle linéaire, alternatif, pour notre problème DARPRel. Nous le notons $P_{MinArretL2}$. Ce second modèle met en jeu les vecteurs inconnus $Z = (Z_{i,k}, i \in D, k = 1..K)$, à valeurs dans $\{0,1\}$ : $Z_{i,k} = 1$ si et seulement si la demande $i$ est traitée par le véhicule $k$, et $T = (T_{k,s}, k = 1..K, s \in 0..S)$, à valeurs dans $\{0,1\}$ : $T_{k,s} = 1$ si et seulement si le véhicule $k$ s'arrête en $s$. Soit $A(s)$ l'ensemble des demandes $i$ telles que $o_i \leq s < s+1 \leq d_i$ et $B(s)$ l'ensemble des demandes $i$ telles que $d_i = s$ ou $o_i = s$. Résoudre le modèle $P_{MinArretL2}$ revient alors à calculer $Z$ et $T$, qui minimisent la quantité "nombre d'arrêts" :

$$\sum_{k \in (1..K)} \sum_{s \in (0..S)} T_{k,s} \tag{7.7}$$

sous les contraintes :

$$\sum_{i \in A(s)} Q_i . Z_{i,k} \ \leq CAP, \forall k = 1..K, \forall s = 0..S-1 \tag{7.8}$$

$$Z_{i,k} \ \leq \ T_{k,s}, \forall k = 1..K, \forall s = 0..S-1, \forall i \in B(s) \tag{7.9}$$

Les contraintes (7.8) expriment le respect de la capacité des véhicules tandis que les contraintes (7.9) sont des contraintes de couplage, qui articulent $Z$ et $T$.

#### Formulation du problème de minimisation des arrêts avec $K$ indéterminé

Les 2 modèles équivalents $P_{MinArretL1}$ et $P_{MinArretL2}$ précédents prennent le nombre de véhicules $K$ comme un paramètre. Si $K$ est libre, c'est-à-dire faisant partie des quantités à déterminer, nous obtenons le modèle suivant, formulé ici de façon non linéaire :

$P_{MinArretL}$ : { Calculer $K$ tel que la valeur optimale de $P_{MinArretL1}(K)$ = valeur optimale de $P_{MinArretL2}(K)$ soit la plus petite possible. }

Si le but est d'abord de déployer le moins possible de véhicules, et à partir de là de minimiser le nombre des arrêts, on obtient :

$P_{MinVehicleArretL}$ : { Calculer la valeur minimale $Kmin$ telle que le programme $P_{MinArretL2(k)}$ a une solution réalisable. }

Il est bien sûr possible de linéariser le programme ainsi formulé. Toutefois, une façon particulièrement bien appropriée de laisser le nombre de véhicules libre dans l'expression de notre problème DARPRel, de DARP avec minimisation des arrêts, va consister à le formuler comme un problème de couverture des demandes par des services :

- on nomme service tout sous-ensemble de $D$ pouvant être traité par un même véhicule, c'est à dire tel que, pour tout $s = 0..S-1$, $\sum_{i \text{ tq } o_i \leq s < d_i} Q_i \leq CAP$ ;
- on note $\Omega$ l'ensemble de tous les services ;
- pour un tel service $\omega \in \Omega$, on note $Ar(\omega)$ l'ensemble des arrêts associés à $\omega$, c'est-à-dire l'ensemble $\cup_{i \in \omega}\{o_i, d_i\}$.

On pose alors le programme $P_{MinArretCol}$ comme suit :

- vecteur inconnu : $X = (X_\omega, \omega \in \Omega)$ à valeur dans $\{0, 1\}$ : $X_\omega = 1$ ssi un des véhicules prend exactement en charge les demandes de $\omega$ ;
- contraintes : pour toute demande $i$, $\sum_{\omega \text{ tq } i \in \omega} X_\omega = 1$ ;
- fonction Objectif : minimiser $\sum_\omega Ar(\omega) X_\omega$.

Le programme ainsi posé met à priori en jeu un vecteur $X$ qui peut être de très grande taille, et on pourra traiter sa relaxation linéaire par génération de colonnes, ce qui explique sa dénomination. Ce dernier modèle présente l'avantage d'une grande flexibilité. Il s'adapte aisément à des modifications d'hypothèses. Ainsi, considérons les cas suivants :

- le nombre de véhicules est limité à $K$ : on rajoute alors la contrainte $\sum_\omega X_\omega \leq K$ ;
- on veut minimiser, non plus le nombre d'arrêts, mais le nombre de véhicules. On écrit alors : minimiser $\sum_\omega X_\omega$.

**Remarque : DARP standard et DARPRel.** Le problème DARPRel dont nous venons de présenter différentes formulations linéaires, est clairement un cas particulier du problème DARP Standard du chapitre 3. Il suffit, pour s'en rendre compte, de préciser comment le réseau virtuel $X$ et la matrice de distances $DIST$ se déduisent des données précédentes :

- le *Depot* se confond avec les stations 0 et S ;
- $X = \cup_{i \in D}\{o_i, d_i\} \cup \{Depot\}$
- définition de $DIST$, pour tout $i \in D$ :
  - $DIST(Depot, o_i) = 1$ et $DIST(Depot, d_i) = +\infty$ ;
  - $DIST(d_i, Depot) = 0$ et $DIST(o_i, Depot) = +\infty$ ;
  - pour tout $i$, $j \in D$ :
    - si $o_i < o_j$ alors $DIST(o_i, o_j) = 1$,
    - si $o_i = o_j$ alors $DIST(o_i, o_j) = 0$,
    - si $o_i > o_j$ alors $DIST(o_i, o_j) = +\infty$,
    - si $d_i < d_j$ alors $DIST(d_i, d_j) = 1$,
    - si $d_i = d_j$ alors $DIST(d_i, d_j) = 0$,
    - si $d_i > d_j$ alors $DIST(d_i, d_j) = +\infty$,

— si $o_i < d_j$ alors $DIST(o_i, d_j) = 1$ et $DIST(d_j, o_i) = +\infty$ ;
— si $o_i = d_j$ alors $DIST(o_i, d_j) = DIST(d_j, o_i) = 0$,
— si $o_i > d_j$ alors $DIST(o_i, d_j) = +\infty$ et $DIST(d_j, o_i) = 1$.

Trivialement, les fenêtres de temps $F(o_i)$, $F(d_i)$, $i \in D$, sont alors mises à $[0\,;\,+\infty]$ et les bornes $\Delta_i$, $\Delta$ sont mises à $+\infty$.

**Remarque : analogie avec les jeux de type *casse-brique***

La minimisation du nombre d'arrêts sur une topologie en ligne peut s'interpréter comme un jeu de type *casse-brique*, comme par exemple le Tetris qui fait partie des problèmes NP-Complet [Demaine et al. (2003)]. Considérons des chargements unitaires (pour l'exemple) et non divisibles (1 unité par demande) et la capacité d'un véhicule également entière. Si nous modélisons le problème comme un jeu (cf. en figure 7.3), il consisterait à faire descendre des blocs de demandes dans les tournées dans le but de gagner un maximum de points. Ces derniers sont attribués selon la taille $t$ de chaque ligne verticale supérieure à 1 pour un véhicule donné. L'exemple de la figure montre un gain de points lorsqu'on obtient 2 lignes verticales de taille 2 (une de chaque côté) pour un total de 2 points. Si le troisième bloc descend dans le but de former trois couches identiques, nous obtenons un point à chaque couple de lignes verticales possible. Autrement dit le nombre de combinaisons de ces couples rapporte 6 points (2*3). Si la capacité était de 4 et qu'un quatrième bloc se superposait, le résultat serait alors de 12 points (2*6). La difficulté est concentrée sur l'ordonnancement de l'insertion des demandes qui sont connues à l'avance, soit très proche du problème de la minimisation des arrêts sur une topologie en ligne. Le calcul des points pourrait très bien s'accorder avec notre problème, en considérant que l'on cherche à minimiser les points qui comptabiliseraient le nombre d'arrêts.

Dans le contexte dynamique d'une instance d'un problème de la minimisation des arrêts, ces demandes peuvent arriver par groupes, comme pour les séries de formes à assembler dans beaucoup de jeux de type *casse-brique* comme le Tetris.

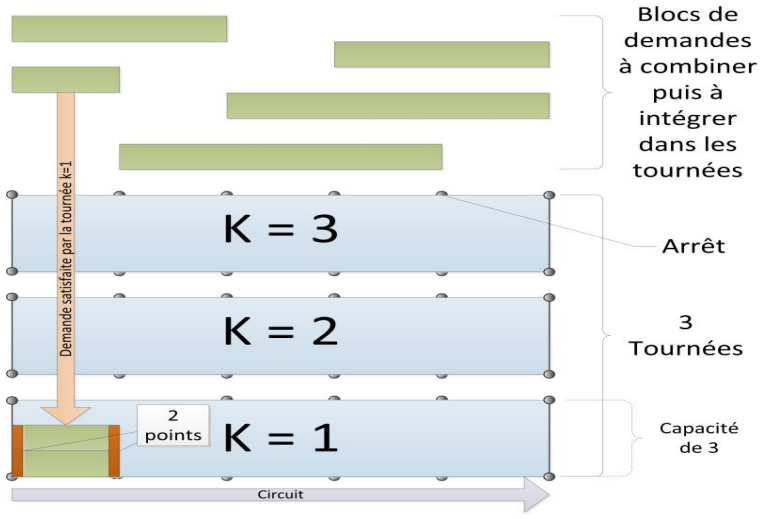


FIGURE 7.3 – Jeu de la minimisation du nombre d'arrêt

### 7.1.2 Modèles sur topologie Circuit

Nous supposons à présent que les véhicules se déplacent le long d'un circuit conforme à la figure 7.1, qu'ils partent d'un nœud spécifique *Depot*, et qu'un véhicule puisse effectuer plusieurs tours avant de commencer à servir une demande. Nous notons $C$ le nombre maximum de tours que chaque véhicule peut effectuer. Le problème se modélise alors selon la même logique que dans le cas dit "en ligne" :

**Modèle** $P_{MinArretC1}$

Soit les vecteurs inconnus suivants :

— $Z = (Z^c_{i,k}, i = 1..|D|, k = 1..K, c = 1..C)$ : $Z^c_{i,k} = 1$ ssi le véhicule $k$ sert la demande $i$ lors de son tour $c$ ;
— $T = (T^c_{k,s}, k = 1..K, c = 1..C, s = 0..S-1)$ : $T^c_{k,s} = 1$ ssi le véhicule $k$ s'arrête sur la station $s$ lors de son tour $c$ ;
— $C = (ChT^c_{k,s}, k = 1..K, c = 1..C, s = 0..S-1)$ : $ChT^c_{k,s}$ représente la charge portée par le véhicule $k$, quand il dépasse la station $s$ au tour $c$.

Nous cherchons à minimiser la fonction Objectif $f$ suivante :

$$f = \sum_{k \in K} \sum_{s=0}^{S-1} \sum_{c=1}^{C} T^c_{k,s} \tag{7.10}$$

sous les contraintes :

$$\sum_{k \in K} \sum_{c=1}^{C} Z^c_{i,k} = 1, \forall\ i \in D \tag{7.11}$$

$$\sum_{k \in K} Z^C_{i,k} = 0, \forall\ i \in D : d_i < o_i \tag{7.12}$$

$$Z^c_{i,k} \leq T^c_{k,o_i}, \forall\ i \in D,\ c = 1..C,\ k \in K \tag{7.13}$$

$$Z^c_{i,k} \leq T^c_{k,d_i}, \forall\ i \in D : d_i > o_i,\ c = 1..C,\ k \in K \tag{7.14}$$

$$Z^c_{i,k} \leq T^{c+1}_{k,d_i}, \forall\ i \in D : d_i < o_i,\ c = 1..C-1,\ k \in K \tag{7.15}$$

$$ChT^1_{0,k} = \sum_{i \in D : o_i = 0} Q_i Z^1_{i,k}, \forall k \in K \tag{7.16}$$

$$ChT^c_{s,k} = ChT^c_{s-1,k} + \sum_{\substack{i\in D:\\ o_i=s}} Q_i Z^c_{i,k} - \sum_{\substack{i\in D:\\ d_i=s,\\ o_i<d_i}} Q_i Z^c_{i,k} - \sum_{\substack{i\in D:\\ d_i=s,\\ o_i>d_i}} Q_i Z^{c-1}_{i,k}, \tag{7.17}$$
$$\forall\ s = 1..S-1,\ k \in K,\ c = 1..C,\ \text{(NB : si } c = 1, Z^0_{i,k} = 0 \text{ par défaut)}$$

$$ChT^c_{0,k} = ChT^{c-1}_{S-1,k} + \sum_{\substack{i\in D:\\ o_i=0}} Q_i Z^c_{i,k} - \sum_{\substack{i\in D:\\ d_i=0}} Q_i Z^{c-1}_{i,k}, \forall\ c = 2..C,\ k \in K \tag{7.18}$$

$$ChT^c_{s,k} = ChT^c_{s-1,k} + \sum_{i\in D: o_i=s} Q_i Z^c_{i,k} - \sum_{i\in D: d_i=s, d_i>o_i} Q_i Z^c_{i,k} - \sum_{i\in D: d_i=s, d_i<o_i} Q_i Z^{c-1}_{i,k}, \forall\ s = 1..S, c = 2..C,\ k \in K \tag{7.19}$$

$$ChT^c_{s,k} \leq CAP, \forall\ s = 0..S-1,\ c = 1..C,\ k \in K \tag{7.20}$$

Les contraintes (7.11) à (7.20) reprennent donc le format du problème sur circuit en intégrant le nombre de tours en joignant la première station à la dernière (qui est également le dépôt). C'est d'ailleurs cette fusion départ/arrivée qui augmente sensiblement le nombre de ces contraintes. Les (7.11) et (7.12) forcent la demande à être satisfaite une seule fois. Lors du dernier tour, d'index C, la demande ne peut être prise en charge si le dépôt se trouve entre l'origine et la destination. Les trois contraintes (7.13, 7.14 et 7.15) correspondent aux contraintes de couplage et les quatre suivantes (7.16, 7.17, 7.18 et 7.19) à l'équilibre de la charge. La (7.20) est la contrainte de capacité. Remarquons enfin que le nombre d'arrêts de la fonction Objectif $f$ (7.10) pourra être pondéré de telle sorte que les demandes doivent être satisfaites le plus tôt possible (avec par exemple un poids de $(1 + c/C)$).

On pourra remarquer :

**Proposition 1**

Dans le cas général, le nombre minimum d'arrêts à une station donnée $s$ est supérieur ou égal à $MAX(\lceil \sum_{o_i=s, i\in D} Q_i/CAP \rceil\ ;\ \lceil \sum_{d_i=s, i\in D} Q_i/CAP \rceil)$.

Preuve. Le nombre d'arrêts nécessaires en station $s$ est comparable au nombre d'opérations à effectuer sur celles-ci. Pour une même demande $i \in D$ donnée, les opérations de chargement en $o_i$ ou de déchargement en $d_i$ ne peuvent s'effectuer sur un même arrêt puisque $d_i \neq o_i$. Au contraire, tous les arrêts de type déchargement permettent autant de chargements sur ce même arrêt et inversement. La capacité étant de $CAP$ nous obtenons le résultat énoncé. Fin Preuve.

**DARPRel sur topologie circuit et DARP standard**

Là encore, il est possible de "plonger" le modèle DARPRel, défini par une topologie en anneau et un nombre limite de tours, dans le formalisme du DARP standard, adapté de la façon suivante :

— l'ensemble $X$ des nœuds du réseau virtuel est l'ensemble $\cup_{i\in D}\{o_i, d_i\}$ des origines/destinations des demandes, considérés comme étant 2 à 2 disjoints, augmenté d'un nœud $DepotD$ (départ) et d'un nœud $DepotD$ (arrivée).
— pour tout arc $(x, y)$ de ce réseau virtuel, nous sommes alors amenés à définir 2 mesures de longueur, que nous noterons DIST et DIST1, et qui vont nous aider, pour la première, à compter le nombre d'arrêts, et pour l'autre, à compter le nombre de tours :

  — DIST($DepotD$, $o_i$) = 1 et DIST1($DepotD$, $o_i$) = 0,
  — DIST($DepotD$, $d_i$) = DIST1($DepotD$, $d_i$) = $+\infty$,
  — DIST($o_i$, $DepotA$) = DIST1($o_i$, $DepotA$) = $+\infty$,
  — DIST($d_i$, $DepotA$) = 0 et DIST1($d_i$, $DepotA$) = 1 si $d_i \neq 0$, 0 sinon,
  — DIST($o_i$, $o_j$) = DIST1($o_i$, $o_j$) = 0 dans le cas où $o_i = o_j$,
  — DIST($d_i$, $d_j$) = DIST1($d_i$, $d_j$) = 0 dans le cas où $d_i = d_j$,
  — DIST($o_i$, $d_j$) = DIST1($o_i$, $d_j$) = DIST($d_j$, $o_i$) = DIST1($d_j$, $o_i$) = 0 dans le cas où $o_i = d_j$,
  — DIST($o_i$, $o_j$) = 1 et DIST1($o_i$, $o_j$) = 0 dans le cas où $o_i < o_j$,
  — DIST($o_i$, $o_j$) = 1 et DIST1($o_i$, $o_j$) = 1 dans le cas où $o_i > o_j$,
  — DIST($d_i$, $d_j$) = 1 et DIST1($d_i$, $d_j$) = 0 dans le cas où $d_i < d_j$,
  — DIST($d_i$, $d_j$) = 1 et DIST1($d_i$, $d_j$) = 1 dans le cas où $d_i > d_j$,
  — DIST($o_i$, $d_j$) = 1, DIST1($o_i$, $d_j$) = 0, DIST($d_j$, $o_i$) = 1 et DIST1($d_j$, $o_i$) = 1 dans le cas où $o_i < d_j$,
  — DIST($o_i$, $d_j$) = 1, DIST1($o_i$, $d_j$) = 1, DIST($d_j$, $o_i$) = 1 et DIST1($d_j$, $o_i$) = 0 dans le cas où $o_i > d_j$.

On vérifie alors aisément que résoudre notre DARPRel sur topologie en anneau, avec une limite $C$ sur le nombre de tours effectués par chaque véhicule revient à résoudre le DARP Standard associé à $X$ et DIST, avec comme objectif de minimiser la somme, ou sens de DIST, des distances parcourues par les véhicules et en respectant la contrainte additionnelle associée à DIST1 :

— pour chaque véhicule, la distance qu'il parcourt au sens de DIST1 ne doit pas excéder $C$.

**Remarque.** La validité de ce qui précède tient au fait qu'un véhicule impliqué dans une solution du modèle DARPRel n'effectuera pas de tour complet sans procéder à au moins une opération de chargement/déchargement.

**Prise en compte des contraintes de temps.**

Si l'on décide à présent d'intégrer, dans les contraintes de notre modèle DARPRel, des contraintes de fenêtres de temps (sur les $o_i$, $d_i$ ou sur les temps de connexion ($T_{Ride}$) de $o_i$ à $d_i$), on voit que l'on est amené à introduire un temps de service additionnel $\delta$ associé aux arrêts.

Le temps nécessaire pour le passage d'un nœud $x \in X$ vers un nœud $y \in X$ sera alors :

— $\delta$DIST$(x, y)$+DIST2$(x, y)$ où DIST2 exprime le temps nécessaire pour aller de $x$ à $y$ en suivant notre circuit sans effectuer de tour additionnel.

On voit que les algorithmes généraux par insertions successives développés dans les chapitres antérieurs s'appliquent alors sans difficulté. Cependant, les modèles DARPRel en "ligne" et en "anneau" sont des cas particuliers du DARP Standard.

Caractérisés à la fois par le fait que les routages sont prédéterminés et par la spécificité des matrices de distances, nous allons proposer pour ces deux modules des algorithmes ad hoc, exacts pour les uns et destinés à nous permettre de disposer de *benchmarks* et approchés pour les autres, et ayant vocation à s'adapter de façon naturelle à des contextes *on line*. Nous remarquons dans le même temps que le modèle "en ligne" est en fait un cas particulier du modèle "en anneau".

## 7.2 Procédés de résolution

### 7.2.1 Une méthode exacte sur topologie Ligne

On applique un schéma classique de *Branch-and-Bound*, basé sur :

- au niveau *Bound*, la résolution de la relaxation fractionnaire $P_{MinArretColLP}$ de $P_{MinArretCol}$ : cette résolution se fait donc par génération de colonnes. Une solution $X$, dérivée d'un ensemble de colonnes actives $\Omega_{Act}$, et induisant une solution duale associée $(\mu, \lambda)$, sera optimale s'il n'existe pas de service $\omega$ dans $\Omega$–$\Omega_{Act}$ tel que : $\mu + \sum_i \lambda > Ar(\omega)$.
- au niveau *Branch* : soit $X$ une solution de $P_{MinArretColLP}$ ; on cherche alors (on vérifie aisément qu'elles doivent exister) 2 demandes $i_1$ et $i_2$ telles que :
  - il existe $\omega$ contenant à la fois $i_1$ et $i_2$ et tel que $X_\omega$ est non nul et fractionnaire ;
  - il existe $\omega'$ qui contient $i_1$ et pas $i_2$ et tel que $X_\omega$ est non nul et fractionnaire. On sépare alors le programme $P_{MinArretCol}$ en :
    - $P_{MinArretColEx}(i_1,i_2)$ : $i_1$ et $i_2$ ne peuvent être dans un même service $\omega$, c.a.d, $X_t = 0$ pour tout $t$ contenant à la fois $i_1$ et $i_2$ (au sens large) ;
    - $P_{MinArretColEt}(i_1,i_2)$ : $i_1$ et $i_2$ doivent être dans un même service, c.a.d, $X_s = 0$ pour tout $\omega$ contenant une de ces 2 demandes et pas l'autre.

On observe que les sous-problèmes susceptibles d'apparaître dans l'arbre d'exploration induit par le processus de *Branch-and-Bound* sont des restrictions de $P_{MinArretCol}$ dans lesquelles certaines colonnes sont en fait interdites. A un nœud $\sigma$ de cet arbre, correspond donc un ensemble de colonnes (services) actifs $Act(\sigma)$ et la valeur courante du *Bound* est donc la valeur optimale de la restriction à $Act(\sigma)$ du programme linéaire $P_{MinArretColLP}$. Il est important de constater que cette restriction, ainsi que les programmes linéaires de base $P_{MinArretCol}$ et $P_{MinArretColLP}$, peuvent ne pas avoir de solution réalisable. De ce fait, la résolution du programme $P_{MinArretColLP}$ doit s'effectuer en 2 temps :

- phase 1 : recherche d'une solution réalisable par la résolution du programme suivant :
  - vecteurs inconnus :$(X_\omega, \omega \in \Omega) \geq 0$ , $Y = (Y_i, i \in D), T \geq 0$ ;
  - contraintes :
    - pour toute demande $i$, $\sum_{\omega \text{ tels que } i \in \omega} X_\omega + Y_i = 1$ ;
    - $T + \sum_\omega X_\omega = K$ ;
  - objectif : minimiser $\sum_i Y_i$ (Au cours de cette phase, des colonnes sont générées, dérivées d'un couple dual $(\mu, \lambda)$, à partir duquel on cherche un service $\omega$ dans $\Omega$-$\Omega_{Act}$ tel que : $\mu + \sum_i \lambda > 0$).
- phase 2 : résolution optimale du programme $P_{MinArretColLP}$, à partir de la solution (base réalisable) obtenue à l'issue de la phase 1.

Comme énoncé plus haut, il sera intéressant de comparer la valeur optimale $K_{Opti}$ du problème portant sur la minimisation de $K$ (extension du problème de coloration) avec le nombre $K^*$ de véhicules associés à la solution optimale du programme

$P_{MinArretCol}$ relâché de la contrainte $\sum_{\omega} X_{\omega} \leq K$.

**Sous-problème de recherche de colonnes : programme PLNE et approches heuristiques**

Le couple dual ($\mu$, $\lambda$) étant donné, ce problème consiste à sélectionner une famille de demandes $Dem \subseteq D$, telle que :

- pour tout $s$ dans 0..S-1, $\sum_{f \in A(s)} Q_f \leq CAP$ ;
- la quantité $\sum_{f \in Dem} \lambda_f + \mu - Ar(Dem)$ est strictement positive ;
- $Dem$ ne figure pas dans un ensemble $\Omega$ -*Interdit*, défini de façon implicite, par une liste de conjonctions ET($i_1$, $i_2$) ou des exclusions EX($i_1$,$i_2$).

Par extension, la contrainte "la quantité $\sum_{f \in Dem} \lambda_f + \mu - Card(Arrêt(Dem))$ est strictement positive" peut être remplacée par un objectif de maximisation de la quantité : $\sum_{f \in Dem} \lambda_f - Ar(Dem)$.

**Remarque :** Dans le cas où l'on se concentre sur la recherche d'une solution réalisable du programme $P_{MinArretCol}$, le terme $Ar(Dem)$ doit être remplacé par 1, et donc doit être supprimé si l'on se réfère à la version "optimisation" de notre problème.

On peut formuler ce dernier problème sous forme PLNE que l'on nommera standard :

**Formulation PLNE Standard** $P\text{-}COL$ :
$z = (z_f, f$ dans D) à valeurs en {0, 1}, $T = (T_s, s = 0..S)$ à valeurs dans {0, 1}, tels que :

- contraintes :
  - pour tout $s = 0..S-1$, $\sum_f Q_f.z_f \leq CAP$ ;
  - pour tout $s = 0..S$, $f$ tel que $s$ dans $Arrêt(F\text{-}Dem)$, $z_f \leq T_s$ ;
  - $Dem$ n'est pas dans la liste Ω-Interdit :
    - pour toute contrainte EX($i_1$,$i_2$) dans S-*Interdit* : $z_{i_1} + z_{i_2} \leq 1$ ;
    - pour toute contrainte ET($i_1$,$i_2$) dans S-*Interdit* : $z_{i_1} - z_{i_2} = 0$ ;
- objectif : maximiser $\sum_{f \in F\text{-}Dem} \lambda_f.z_f - \sum_{s=0..S} T_s$.

Ce problème peut se traiter de plusieurs façons :

- par une recherche rapide via un algorithme glouton ;
- par une recherche exacte (ou *presque exacte*) dans l'hypothèse où la précédente approche échoue. Cette dernière est réalisée par une application *CPLEX* au programme PLNE $P\text{-}COL$ ou alors un algorithme ad hoc.

En fait, la recherche rapide via un algorithme glouton peut s'envisager selon 2 approches données page suivante.

**Approche 1 : par balayage de l'espace des demandes.**

Le principe est simple : on balaye l'espace des demandes et on prend à chaque fois la décision d'insérer ($OUI$) ou pas ($NON$) la demande.

*Différenciation du OUI et du NON.*

L'algorithme peut bien sûr être randomisé en jouant sur l'ordre selon lequel les demandes sont traitées. Il peut aussi l'être en randomisant la réponse $OUI/NON$, à chaque étape de traitement d'une demande. On peut enfin faire varier aléatoirement les coefficients multicritères.

*Evaluation de la réponse OUI/NON.*

Une demande $i$ ne pourra être éventuellement insérée que si son insertion n'implique pas de violation de la contrainte de capacité. La réponse à l'insertion de $i$ sera d'autant plus facilement OUI que :

- la durée nécessaire à la prise en charge de la demande $i$ est petite ;
- sa charge $Q_i$ est de petite valeur ;
- le poids $\lambda_i$ est grand ;
- le nombre $Ar\text{-}aux(i,\ D\text{-}Ins)$ d'arrêts nouveaux créés par l'insertion de $i$ est petit (il est à priori égal à 0, 1 ou 2).

Il peut aussi être prolongé en un algorithme d'exploration arborescente filtrée en créant, pour certaines demandes "sensibles" (celles pour lesquelles l'écart entre $OUI$ et $NON$ a été le plus faible), un nœud de *backtracking*.

**Approche 2 : par choix successifs de la demande à insérer.**

On procède par insertions successives des demandes dans $Dem$, jusqu'à un signal d'arrêt. A chaque étape, on évalue la qualité des demandes non insérées, et on choisit celle dont la valeur est optimale et au-delà d'un certain seuil.

*Evaluation de la qualité d'une demande à insérer.*

Soit $D\text{-}Ins$ l'ensemble des demandes déjà insérées, et $i$ une demande non encore insérée. Cette demande $i$ ne pourra être éventuellement insérée que si la contrainte de capacité est respectée. Dans ce cas, la qualité $\text{Qual}(i,\ D\text{-}Ins)$ sera d'autant plus grande que la durée de la demande $i$ est courte, que sa charge $Q_i$ est petite, que le poids $\lambda_i$ est grand et que le nombre $Ar\text{-}aux(i,\ D\text{-}Ins)$ d'arrêts nouveaux créés par l'insertion de $i$ est faible.

La quantité $\sum_{f \in D\text{-}Ins} \lambda_f - Ar(D\text{-}Ins)$ n'étant pas forcément monotone au fur et à mesure que les insertions se font, il faut mémoriser le meilleur $D\text{-}Ins$, qui constitue alors le résultat $Dem$ final.

L'algorithme peut être aisément *randomisé* en choisissant la demande insérée, non plus de façon déterministe en prenant celle de qualité maximale, mais de façon aléatoire, en sélectionnant les 2 ou 3 demandes de qualité maximale et en tirant au hasard parmi celles-ci. Il peut aussi être randomisé en faisant varier aléatoirement les coefficients multicritères.

**La recherche exacte via un algorithme ad hoc**

La recherche exacte s'effectue au moyen de l'application d'un procédé de décomposition lagrangienne à un modèle orienté Flot. Rappelons que la version "optimisation" du problème de la Colonne consiste à sélectionner une famille de demandes $Dem \subseteq D$, telle que :

— pour tout $s$ dans 0..S-1, $\sum_{f \in A(s)} Q_f \leq CAP$ ;
— la quantité $\sum_{f \in Dem} \lambda_f - Ar(Dem)$ est maximale ;
— $Dem$ ne figure pas dans un ensemble $\Omega$ -*Interdit*, défini de façon implicite, par une liste de conjonctions ET($i_1$,$i_2$) ou des exclusions EX($i_1$,$i_2$).

On peut donner à ce problème une formulation de type "Flot", en s'appuyant sur le *Réseau des Demandes*, défini comme suit :

— les nœuds de ce réseau sont les entiers dans {0..S} ;
— les arcs de ce réseau sont :
  — les arcs *chaîne* $e_s$ = [s, s+1] de capacités 0 et $+\infty$ et de coût nul ;
  — les arcs *demande* $a_i = [o_i, d_i]$ de capacités 0 et $Q_i$, de coût $c_i$ ;
  — l'arc *retour* [S, 0], de capacités [$CAP$,$CAP$] et de coût nul.

**Formulation Flot** {$F$ flot sur $H$, entier, $T = (T_s,\ s = 0..S)$ à valeurs dans {0, 1}, tels que :

— pour chaque arc $e$ de $H$, de type *Demande*, $F(e)$ vaut 0 ou $Q_e$ ; (C1)
— $F$ est compatible avec les capacités $Min$ et $Max$ ; (C2)
— pour toute conjonction EX($i_1$,$i_2$), la somme $F(a_{i_1})/Q_{i_1} + F(a_{i_2})/Q_{i_2}$ des valeurs de flot sur les arcs *demande* associés à $i_1$ et $i_2$, doit être au plus égale à 1 ; (C3)
— pour toute exclusion ET($i_1$,$i_2$), les valeurs de flot sur les arcs *demande* associés à $i_1$ et $i_2$, doivent être égales ; (C4)
— pour tout $s = 0..S$, $i$ tel que $o_i = i$ ou $d_i = i$, $F(e_i) \leq Q_i.T_s$ ; (C5)
— Max = $\lambda$.F - **1**.T ;}

On s'intéresse au traitement de ce problème via une relaxation lagrangienne des contraintes (C1, C3, C4, C5).

***Relaxation Lagrangienne de la Formulation Flot***.

Soit $k = (k_{s,i},\ s = 0..S,\ i$ tel que $s$ dans Arrêt($i$)), $h$-$ET$ = (h-$ET_{i_1,i_2}$, $i_1$, $i_2$ tels que $ET(i_1,\ i_2)$), $h$-$EX$ = (h-$EX_{i_1,i_2}$,$i_1$, $i_2$ tels que $EX(i_1,\ i_2)$) de signe quelconque, un vecteur de Lagrange. On peut poser le lagrangien $Lag(F, T, k, h\text{-}ET, h\text{-}EX)$ (7.21).

$$
\begin{aligned}
Lag(F, T, k, h\text{-}ET, h\text{-}EX) = \lambda.F - 1.T &- \sum_{s,i} k_{s,i}.(F(e_i) - Q_i.T_s) \\
&- \sum_{i_1,i_2} h\text{-}EX_{i_1,i_2}.(F(a_{i_1})/Q_{i_1} + F(a_{i_2})/Q_{i_2} - 1) \\
&- \sum_{i_1,i_2} h\text{-}ET_{i_1,i_2}.(F(a_{i_1})/Q_{i_1} - F(a_{i_2})/Q_{i_2})
\end{aligned}
\tag{7.21}
$$

**Remarque** : Dans cette notation simplifiée, il y a symétrie dans les rôles respectifs de $i_1$ et $i_2$. On considère donc que ET($i_2$,$i_1$) et ET($i_1$,$i_2$) recouvrent la même contrainte (même considération pour EX($i_1$, $i_2$)).

La maximisation de $Lag(F, T, k, h\text{-}ET, h\text{-}EX)$ pour $F$, $T$ soumis à (C1, C2) se sépare alors en 2 sous-problèmes :

**Problème en** $T$ : { $T = (T_s,\ s = 0..S)$ à valeurs dans {0, 1}, qui maximise $\sum_{s,i} k_{s,i}.Q_i.T_s - \mathbf{1}.T = \sum_s T_s(\sum_{i/s\in Arrêt(i)} k_{s,i}Q_i - 1)$. La solution de ce problème est triviale : $T_s = 1$ si $1 \leq \sum_{s,i} k_{s,i}Q_i$ et $T_s = 0$ sinon. On note $V_k$ sa valeur optimale.}

**Problème en** $F$ : {Flot $F$ entier, compatible avec les capacités $Min$ et $Max$, tel que pour chaque arc $e$ de $H$, de type *demande*, $F(e)$ vaut 0 où $Max(e)$ est donnée en formule (7.22) ; }

(7.22)

$$Max(e) = \lambda.F - \sum_{s,i} k_{s,i}(F(a_i)) - \sum_{i_1,i_2} h\text{-}EX_{i_1,i_2}(F(a_{i_1})/Q_{i_1} + F(a_{i_2})/Q_{i_2} - 1) - \sum_{i_1,i_2} h\text{-}ET_{i_1,i_2}(F(a_{i_1})/Q_{i_1} - F(a_{i_2})/Q_{i_2})$$

On note $W^*_{k,h\text{-}ET,h\text{-}EX}$ sa valeur optimale. On note *Problème Simple en F* le problème en F relâché de la contrainte (C1) et on note $W^*_{k,h\text{-}ET,h\text{-}EX}$ la valeur optimale de ce dernier problème. On sait que pour tout vecteur $k$, $h\text{-}ET$, $h\text{-}EX$ la valeur optimale du problème de la colonne est majorée par la valeur $V_k + W_{k,h\text{-}ET,h\text{-}EX} \leq V_k + W^*_{k,h\text{-}ET,h\text{-}EX}$. Dans le cas où les demandes sont unitaires, c'est-à-dire où les valeurs $Q_f$ sont égales à 1, alors $W_{k,h\text{-}ET,h\text{-}EX} = W^*_{k,h\text{-}ET,h\text{-}EX}$.

Une solution ($F$, $T$) de ces 2 problèmes étant calculée, le sous-gradient $\Delta = \Delta_{F,T}$ de $Lag(F, T, k, h\text{-}ET, h\text{-}EX)$ s'écrit :

$\Delta = \Delta_{F,T}$ = ($F(a_i)$ - $Q_i.T_s$ , $s = 0..S$, $i$ tel que $s$ dans Arrêt($i$)), ($F(a_{i_1})/Q_{i_1} + F(a_{i_2})/Q_{i_2} - 1$, $i_1$, $i_2$ tels que $ET(i_1, i_2)$), ($F(a_{i_1})/Q_{i_1}$ - $F(a_{i_2})/Q_{i_2}$, $i_1$, $i_2$ tels que $EX(i_1, i_2)$).

On est alors en mesure de traiter de façon exacte notre problème de recherche de colonnes à l'aide d'un processus de *Branch and Bound*.

N'importe quelle valeur $W^*_{k,h\text{-}ET,h\text{-}EX}$ fournit une borne supérieure pour le problème de la colonne. Le processus de stérilisation induit une dérive du calcul des objets $F$ et $T$ associés à la valeur $W^*_{k,h\text{-}ET,h\text{-}EX}$ considérée : on teste si $F$ et $T$ ainsi dérivés sont des solutions réalisables du problème.

Le processus de branchement est associé au vecteur $T$ : un nœud de branchement en considérant, dans cet ordre de priorité, une violation de contrainte :

- s'il s'agit d'une contrainte de type (C5), s dans {0..S}, on distingue les 2 alternatives $T_s = 1$ OU $T_s = 0$ ;
- s'il s'agit d'une contrainte de type (C1), associée à une demande $i$, on distingue les 2 alternatives : $i \in Dem$ OU $i \notin Dem$ ;
- s'il s'agit d'une contrainte de type (C3), associée à 2 demandes $i_1$ et $i_2$, on distingue les 2 alternatives : $i_1 \in Dem$ (et donc $i_2 \notin Dem$) OU $i_1 \notin Dem$ ;
- s'il s'agit d'une contrainte de type (C4), associée à 2 demandes $i_1$ et $i_2$, on distingue les 2 alternatives : $i_1 \in$ Dem (et donc $i_2 \in Dem$) OU $i_1 \notin Dem$ (et donc $i_2 \notin Dem$).

### 7.2.2 Deux approches par insertions sur topologie Ligne

Notre problème ayant vocation à être traité, d'un point de vue applicatif, en temps réel, nous proposons à présent 2 heuristiques d'insertions, spécifiquement conçues pour le modèle DARPRel.

**Heuristique 1 - Insertions successives aiguillées par courbe de profil.** On insère les demandes les unes après les autres dans les véhicules. A un instant donné en cours de processus, la situation est donc la suivante :

- certaines demandes ont été insérées dans les véhicules, définissant un ensemble $D\text{-}Ins$ ;
- pour chaque véhicule $k$ dans $1..K$, on dispose d'une courbe profil $P_k$ (la courbe qui, à chaque intervalle $[s,\ s+1]$ associe la place libre dans $k$ durant cet intervalle et l'ensemble des arrêts de $k$) ;
- pour chaque demande $i$ non encore insérée, on dispose de la liste des véhicules encore susceptibles d'accueillir $i$.

La demande cible $i_0$ est alors une des demandes les moins insérables, c'est-à-dire pour lesquelles le nombre de véhicules d'accueil est le plus petit. 3 cas sont pris en compte sur l'idée que l'intégration d'une nouvelle demande crée 0, 1 ou 2 nouveaux arrêt(s) :

- $1^{er}$ cas : S'il existe un véhicule $k_0$ (ou un groupe de véhicules) dont les arrêts incluent les 2 arrêts de $i_0$, alors on insère $i_0$ dans $k_0$ (tiré aléatoirement si plusieurs véhicules correspondent) ;
- $2^{ème}$ cas : Si le premier cas n'est pas satisfait, mais qu'il existe un (ou des) véhicule $k$ partageant un arrêt avec $i_0$, alors on évalue, pour le nouvel arrêt $s_0$ qui serait ainsi créé en $k$ par l'insertion de $i_0$, la marge $O(k,\ s_0)$ de $k$ au voisinage de $s_0$. Cette marge est égale à une intégrale noyau de $P_k$ centrée en $s_0$, c'est-à-dire à l'intégrale d'un produit $H^{s_0}.\ P_k$, où $H$ est une fonction en escalier, croissante à gauche de 0, décroissante à droite de 0, et égale à 1 sur l'intervalle [-1, 1] et $H^{s_0}$ est la translatée de $H$ d'amplitude $s_0$. On choisit alors le véhicule d'insertion $k_0$ de telle sorte qu'il maximise la valeur $O(k,\ s_0)$ ;

— $3^{ème}$ cas : si ni le premier ni le second cas ne sont satisfaits, alors on identifie les deux points d'arrêts $s_1$ et $s_2$ de $i_0$, et on choisit le véhicule d'insertion $k$ de telle sorte qu'il maximise O($k$, $s_1$) + O($k$, $s_2$).

Cet algorithme peut clairement être randomisé en choisissant au hasard, à chaque étape, la demande à insérer, cela tout en respectant la règle de priorité qui fait que la demande à insérer doit être une des plus contraintes.

**Heuristique 2 - Traitement du problème de minimisation des arrêts par balayage et couplage des arrêts.** On procède ici en balayant l'ensemble des arrêts, et, à chaque étape, c'est-à-dire pour chaque point d'arrêt $s$, en répartissant l'ensemble $D_s$ des demandes dont l'origine est en $s$ parmi les $K$ véhicules. A un instant donné en cours de processus, la situation est donc la suivante :

— $s$ est l'arrêt courant : les ensembles $D_0$.. $D_{s-1}$ ont donc été placés dans les véhicules ;
— on dispose, pour chaque véhicule $k$, de la restriction $P\text{-}Rest_k$ à partir de $s$ de la courbe profil $P_k$ ;
— pour chaque véhicule $k$, on connaît ses arrêts $Ar\text{-}cour(k)$ conformément aux insertions déjà réalisées.

On doit alors répartir les demandes de $D_s$ : les répartitions admissibles correspondent clairement à un couplage généralisé de $D_s$ sur $\{1..K\}$. Le modèle **Couplage avec Arrêts** au nœud s pour ce sous-problème se formule comme suit :

Calculer $Z = (Z_{i,k},\ i \in D$ et tels que $o_i = s,\ k = 1..K)$, $T = (T_{r,k},\ r \geq s,\ k = 1..K,$ $r$ non dans $Ar\text{-}cour(k))$, à valeurs en $\{0, 1\}$, tels que :

— pour tout $i$, $\sum_k Z_{i,k} \leq 1$ ;
— pour tout $k$, $\sum_i Q_i Z_{i,k} \leq CH^s_k$, où $CH^s_k$ désigne la place disponible dans le véhicule $k$ quand celui-ci sort de $s$ (soit $CH^s_k = CAP - ChT_{s,k}$) ;
— pour tout $i$, $k$ et $s$ tel que $s \notin Ar\text{-}cour(k)$, $\sum_k Z_{i,k} \leq T_{s,k}$ ;
— pour tout $i$, $r$, $k$, tels que $d_i = r$, $r \notin Ar\text{-}cour(k)$, $Z_{i,k} \leq T_{r,k}$ ;
— $\sum_{k,r} T_{r,k} \leq 1$ est minimal.

Ce modèle PLNE, à priori de petite taille, peut éventuellement être traité directement. Il peut aussi faire l'objet d'un traitement GRASP, qui consiste en l'application plusieurs fois répétée d'un algorithme d'insertion gloutonne randomisée (cf. schéma algorithmique 28), suivie d'un processus de descente (transformation locale). Le processus de transformation locale est basé sur deux opérateurs :

— $Move(i, k, k')$ : prend une demande $i$, affectée à un véhicule $k$, et la déplace sur un véhicule $k'$ ;
— $Exchange(i, i')$ : prend 2 demandes affectées à 2 véhicules différents et les échange.

Ces 2 heuristiques gloutonnes peuvent se prolonger en des schémas GRASP, via une stratégie de Descente qui consiste à identifier un véhicule $k_0$ et un arrêt $s_0$ tels que le nombre de demandes $i$ telles que $o_i = s_0$ ou $d_i = s_0$ et qui sont chargées par $k_0$

est faible. On cherche alors à appliquer les opérateurs *Exchange* et *Move* de façon à vider $k_0$ de ces demandes (application de *Move*). Une boucle intermédiaire est alors mise en œuvre, à l'intérieur de laquelle une demande $i_o$ chargée par $k_0$ et telle que que $o_{i_0} = s_0$ ou $d_{i_0} = s_0$ est alors identifiée comme "cible" à transférer. On applique les opérateurs *Move* et *Exchange* aux autres véhicules et aux autres demandes de façon à augmenter le maximum, pour les véhicules $k \neq k_0$ qui possèdent comme arrêts $o_{i_0}$ et $d_{i_0}$, des valeurs intégrales des fonctions de profil $P_k$ entre $o_{i_0}$ et $d_{i_0}$ sans modifier le coût de base lié aux arrêts.

**Algorithme 28** Schéma d'insertion gloutonne pour le couplage avec arrêts

affecter dans *D-Aux* les demandes $d$ telles que $o_i = s$ ;
$Continu \leftarrow VRAI$ ;
**Tant que** *Continu* et *D-Aux* non vide **Faire**
  on retire aléatoirement une demande $i$ de *D-Aux* ;
  **Si** il existe au moins un $k$ tel que que $o_i$ **et** $d_i$ sont inclus dans $Ar\text{-}Cour(k)$ et que $Q_i <= CH_k$ **Alors**
    choisir (si besoin) $k$ aléatoirement et mettre à jour *D-Aux*, $Ar\text{-}cour(k)$ et $CH_k$ ;
  **Sinon**
    **Si** il existe au moins un $k$ tel que que $o_i$ **ou** $d_i$ est inclus dans $Ar\text{-}Cour(k)$ et que $Q_i <= CH_k$ **Alors**
      choisir (si besoin) $k$ aléatoirement et mettre à jour *D-Aux*, $Ar\text{-}cour(k)$ et $CH_k$ ;
    **Sinon**
      $Continu \leftarrow FAUX$ ;
    **Fin si**
  **Fin si**
**Fin tant que**

### 7.2.3 Problème sur topologie Circuit : adaptation du schéma par insertions successives

Nous nous contentons ici d'expliquer rapidement comment les procédures d'insertions vues dans les chapitres précédents s'adaptent au DARPRel, considéré comme durci par des contraintes de fenêtres de temps, de durée maximum d'acheminement pour chaque demande et de durée maximum pour chaque tournée. Le but est alors de se positionner au plus proche d'une exploitation VIPA réelle suivant son évolution technologique actuelle. L'optimisation de la supervision d'une flotte de ces véhicules autonomes cherche avant tout à minimiser le nombre d'arrêts. On a vu en section 7.1.2 comment le problème ainsi posé s'intégrait dans le *framework* général du DARP. On va expliquer comment la phase de propagation de contraintes peut être simplifiée en tirant partie des spécificités du problème.

**Optimisation de la procédure de propagation de contraintes** Le chapitre 3 introduisait 5 règles d'inférence permettant de statuer la validité d'une nouvelle insertion en termes de satisfaction des contraintes de temps. C'était l'étape la plus chronophage du processus d'insertion. Ici, les modifications apportées au modèle nous permettent d'optimiser cette procédure en posant une première contraction d'une

fenêtre de temps pour l'ensemble des fenêtres d'un même "bloc". Nous nommons ici "bloc", un ensemble de nœuds du réseau virtuel $X$ qui vont correspondre à un même arrêt pour un certain véhicule. Rappelons que l'amplitude $\theta$ des fenêtres de temps à l'origine est la même pour l'ensemble de ces fenêtres et l'amplitude de la fenêtre de destination est la plus grande possible. La figure 7.4 schématise trois fenêtres de temps relatives à trois nœuds $x$, $y$ et $z$ du même bloc et le tableau 7.1 donne les différentes valeurs associées aux 3 fenêtres.

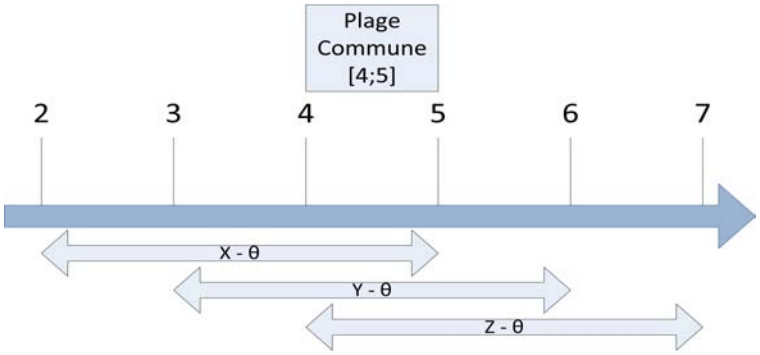


FIGURE 7.4 – Exemple : fenêtres de temps associées à un même label

| | X | Y | Z |
|---|---|---|---|
| F.min | 4 | 3 | 2 |
| F.max | 7 | 6 | 5 |

TABLE 7.1 – Exemple : fenêtres de temps à amplitude constante

Nous voyons qu'il est facile de faire ressortir la plage de temps commune à toutes les fenêtres d'un même arrêt, cette dernière est égale à $[Max_{x \in s}\ F.min(x)\,;Min_{x \in s}\ F.max(x)]$. La propagation peut alors se faire de deux façons :

- l'insertion au sein d'un bloc (d'un même arrêt) est réalisée de façon que les fenêtres de temps soient présentées de la plus tardive à la plus récente. De cette manière les règles d'inférence R1 et R2 contracteront plus rapidement les fenêtres ;
- une procédure contractant directement les fenêtres avant de lancer la fonction *Propage*. Toutes les fenêtres d'un même bloc $s$ sont alors identiques et de la forme $[Max_{x \in s}\ F.min(x)\,;Min_{x \in s}\ F.max(x)]$.

Le second point nous amène à employer le processus de propagation des contraintes directement sur les blocs et non sur l'ensemble des nœuds d'une tournée. Ceci pourrait être le cas pour les deux premières règles. Il est important que les règles R3 et R4 contractent toujours les fenêtres par la durée maximum d'acheminement sur chacun des nœuds d'un même bloc.

Il faut rappeler aussi que la procédure de propagation de contraintes est appelée dès lors que l'on a à tester une insertion d'une demande $i_0 \in D$ paramétrée par $(k, x, y, v)$. Le couple de points d'ancrage $(x, y)$ ($x$ pour l'insertion de l'origine $o_{i_0}$ et

$y$ pour la destination $d_{i_0}$) sont habituellement évalués sur chaque nœud possible si $x \ll_k^= y$. Nous réduisons le nombre de points d'ancrage de façon à n'en considérer qu'un seul par bloc. Ceci est tout à fait possible puisqu'ici les temps de chargement et déchargement ne sont pas associés aux volumes. Ainsi, nous avons mis à jour la procédure *TestInsertion* par *TestInsertLoop* du chapitre 3 où, avant de lancer la propagation de contraintes, nous contractons les fenêtres de la demande à insérer en fonction de celles déjà associées aux deux blocs cibles. *TestInsertLoop* peut se résumer en cinq points (cf. algorithme 29) :

— on insère une demande $i$ au sein d'un bloc si la fenêtre de temps de l'origine de $i$ a une plage commune avec toutes les fenêtres des autres nœuds du même bloc ; (EB1)

— vient ensuite le test sur la charge totale suite à la nouvelle insertion. La procédure *TestChargeLoop* va propager la valeur de la nouvelle charge entre les blocs $b_{o_i}$ et $b_{d_i}$, $b_{d_i}$ non compris ; (EB2)

— on récupère les points d'ancrage associés aux deux blocs. Nous considérons ici qu'au sein d'un même bloc, leur place n'a aucune importance ; (EB3)

— on contracte chaque fenêtre au sein de chaque bloc de façon à ce que toute fenêtre (pré-)propagée $FP$ des nœuds $x$ d'un même bloc $b$ soit incluse dans $FP(x) = [Max_{x \in b} F.min(x); Min_{x \in b} F.max(x)]$ ; (EB4)

— on lance la propagation de contraintes de temps du chapitre 3 et on récupère la différence de coût depuis l'insertion. Les méthodes $EvalTour1$ et $Evaltour2$ du DARP standard peuvent alors être exécutées pour calculer les rendez-vous se positionnant à l'intérieur des fenêtres contractées. C'est la méthode $Eval(\Gamma)$ qui va calculer le nombre d'arrêts en sommant le nombre de blocs dans les tournées de $\Gamma$. (EB5)

**Algorithme 29** TestInsertLoop

**Entrées:** $(\Gamma, b_{o_i}, b_{d_i}, i, \theta)$ ;
**Sorties:** $(InsertPossible, v, x, y)$

$tempMinFMax_{b_{o_i}} \leftarrow Min(F.max(o_i); Min_{y \in b_{o_i}}(F.max(y)))$ ;
$tempMaxFMin_{b_{o_i}} \leftarrow Max(F.min(o_i); Max_{y \in b_{o_i}}(F.min(y)))$ ; (EB1)
**Si** $tempMinFMax_{b_{o_i}} - tempMaxFMin_{b_{o_i}} \geq 0$ **Alors**
  **Si** TestChargeLoop($\Gamma$,$b_{o_i}$,$b_{d_i}$,i) (EB2) **Alors**
    $(x, y) \leftarrow calculPointsAncrage(\Gamma, b_{o_i}, b_{d_i}, i)$ ; (EB3)
    on insère temporaire la demande par $(x, y)$ ; on contracte les fenêtres de temps $FP$ tel que pour chaque nœud $x$ de chaque bloc $b$ de $\Gamma$ : $FP(x)$ est incluse dans $[Max_{y \in b}(F.min(y)); Min_{y \in b}(F.max(y))]$ ; (EB4)
    $InsertPossible \leftarrow Propage(\Gamma, x, y)$ ; (EB5)
    on retire la demande $i$ ;
    **Si** $InsertPossible$ **Alors**
      $v \leftarrow EvalTourMinArrets(\Gamma, FP)[0] - Eval(\Gamma)$ ;
    **Fin si**
  **Fin si**
**Fin si**
**Retourner** $(InsertPossible, v, x, y)$ ;

## 7.3 Résultats expérimentaux

**Objectifs**

Pour le modèle DARPRel "en Ligne", nous avons implémenté le processus par *Branch and Bound* associé à la représentation de notre problème, orienté génération de colonnes, qui nous permet d'obtenir des résultats exacts pour des tailles de problèmes supérieurs à celles que traite CPLEX (version 12) appliqué au modèle $P_{MinArretL2}$. Cette implémentation met en œuvre le *framework* générique $SCIP$, et nous permet d'obtenir des résultats de références pour nos heuristiques. Nous présentons ici des résultats permettant d'évaluer la marge d'erreur induite par l'application de l'heuristique 1 : "Insertions successives et courbes de profils", la plus aisément adaptable à un contexte dynamique.

Pour le modèle DARPRel "sur Circuit", on ne dispose pas de *benchmark*. Dès lors, les tests réalisés cherchent à évaluer les corrélations pouvant exister entre les différentes caractéristiques d'une solution : nombre d'arrêts, nombre de véhicules... Pour cela, on part d'une solution réalisable quelconque, on applique l'algorithme d'insertions décrit en 7.2.3, et on observe l'évolution de ces caractéristiques en cours et au terme de l'exécution de l'algorithme.

**Les instances**

**DARPRel sur Ligne.**

On considère des demandes définies par leur origine, leur destination et leur charge. Ces dernières sont tirées aléatoirement de 1 à $CAP$, chaque véhicule a ici une capacité de 10 et le nombre de demandes générées par instance varie de 30 à 120.

Les instances se caractérisent avec $S$, $|D|$, $S_{act}$, $Q_{moy}$ et $Dist_S$, respectivement le nombre d'arrêts sur la ligne, le nombre de demandes de l'instance, le nombre d'arrêts relatifs aux demandes de $D$, la charge moyenne des éléments de $D$ et le nombre moyen de stations séparant l'origine et la destination d'une demande. Les paramètres d'entrées de 10 instances sont donnés par le tableau 7.2.

| Inst. | $\|D\|$ | $S$ | $S_{act}$ | $Q_{moy}$ | $DIST_S$ |
|---|---|---|---|---|---|
| 1 | 30 | 5 | 5 | 2 | 2.5 |
| 2 | 30 | 10 | 10 | 3 | 5 |
| 3 | 40 | 5 | 5 | 5 | 2.5 |
| 4 | 40 | 10 | 10 | 5 | 5 |
| 5 | 70 | 30 | 28 | 2 | 10 |
| 6 | 70 | 50 | 43 | 2 | 15 |
| 7 | 70 | 100 | 78 | 2 | 30 |
| 8 | 100 | 100 | 89 | 5 | 50 |
| 9 | 120 | 30 | 30 | 5 | 15 |
| 10 | 120 | 50 | 48 | 5 | 25 |

TABLE 7.2 – Paramètres des instances

**DARPRel en Circuit.**

Nous considérons d'abord que les temps nécessaires pour les opérations de chargement/déchargement sont nuls. Nous exploitons un laps de temps de 30 minutes en début de matinée au sein, par exemple, d'un environnement industriel où les allées et venues restent uniformes de la minute 480 jusqu'à la minute 510, soit de 8h à 8h30. Nous avons considéré le cas le plus simple : un cercle centré dans le plan et de rayon 1 où sont disposées les différentes stations de manière équidistante sachant qu'un unique tour mesure $2\pi$ (4 tours par véhicule peuvent donc être envisagés dans [480 ;510]). Les dépôts de départ et d'arrivée des véhicules sont tous réunis au point de coordonnées (1,0). Les distances séparant les origines et les destinations, toutes inférieures à $\Delta^{tour} = 2\pi$, sont calculées de telle façon que les véhicules passent par toutes les stations situées sur le chemin joignant les deux nœuds. La génération de la date à laquelle chaque usager envisage un départ, se fait de manière uniforme sur $[480\,;\, 510 - \Delta^{tour}]$ et permet de créer chacune des bornes $\Delta_i, \forall i \in D$ telle que $\Delta_i = \eta\Delta^{tour}, \forall i \in D$ et $\eta$ étant un paramètre tiré aléatoirement entre deux bornes fixées selon le type d'instance créée. Pour ces tests, nous interdisons aux véhicules de réaliser plusieurs tours pour une même demande : $\eta = 1$. Nous générons ensuite un second groupe d'instances à partir du même protocole (et toujours avec des temps de service nuls). Elles se différencient par leur nombre de demandes |D| : 12, 24 et 48. Puis, à partir de ces trois instances, dont chaque demande a une fenêtre de temps d'amplitude 5 à l'origine, nous en construisons encore trois nouvelles en les modifiant uniquement par un élargissement des fenêtres allant jusqu'à une amplitude de 10 (les 5 minutes supplémentaires englobent uniformément les premières).

Dans un deuxième temps, on considère que les temps de service (opérations de chargement/déchargement) sont non nuls et constants, au sens où, pour une station d'arrêt $x$, toutes les opérations de chargement/déchargement réalisées en $x$ le seront dans une durée $\delta$, l'instance demeurant, pour ses autres caractéristiques, définies comme précédemment. Le tableau 7.3 de la page suivante nous donne les différents paramètres des instances.

Ces instances sont utilisées à deux reprises avec une variation sur la durée maximum d'acheminement - ang. *maximum ride time* - $\Delta_i$. Pour le premier ensemble, il faut noter que de $\Delta_1$ pour toute demande $i$ de $D$ n'est égal qu'à $2\pi + 1$, ce qui empêche les véhicules de s'arrêter à chaque nœud. En effet, avec l'objectif de minimiser les arrêts, des durées $\Delta_i$ plus larges permettent aux véhicules de s'arrêter à chaque station, ce qui peut s'avérer désagréable pour les usagers. Nous nous limitons ici à un temps légèrement supérieur à celui de la réalisation de toute la boucle sans aucun arrêt.

| Inst. | $\|D\|$ | $K_{Dispo}$ | $CAP$ | $\theta$ | $\delta$ | $S$ | $\Delta_1$ | $\Delta_2$ |
|---|---|---|---|---|---|---|---|---|
| $D60$ | 60 | 10 | | | | | | |
| $D90$ | 90 | 15 | | | | | | |
| $D120$ | 120 | 20 | | | | | | |
| $D150$ | 150 | 25 | | | | | | |
| $D180$ | 180 | 30 | 6 | 10 | 1 | 9 | $2\pi+\delta$ | $2\pi+S\delta$ |
| $D216$ | 216 | 36 | | | | | | |
| $D252$ | 252 | 42 | | | | | | |
| $D288$ | 288 | 48 | | | | | | |

TABLE 7.3 – Paramètres des instances

**Résultats et analyse**

**DARPRel en Ligne.**

Notons $K_{Min}$ le nombre minimal de véhicules obtenu par la résolution des problèmes *MinNbVehicules* et $K_{opti}$ celui de *MinArret*. Les nombre d'arrêts optimaux respectifs sont notés $f_{min}$ et $f_{opti}$. Les notations $nbN$, $Col$ et $LB$ sont respectivement le nombre de nœuds induits par l'arbre de recherche, le nombre de colonnes générées et le nombre d'arrêts obtenus par relaxation linéaire. $CPU$ est le temps d'exécution en secondes du processus entier. En ce qui concerne l'heuristique, $K_{ins}$, $Gap_f$, $CPU_{ins}$ et $R$ sont respectivement la population de la flotte, le pourcentage $Gap$ entre le nombre d'arrêts trouvé par l'heuristique et l'optimale, le temps d'exécution en secondes $CPU$ et le nombre de réplications dont les résultats témoignent.

L'étude du tableau 7.4, nous permet de constater que la borne LB donnée par la relaxation continue est particulièrement proche du problème en nombres entiers.

| Inst. | $K_{opti}$ | $f_{Opti}$ | $K_{Min}$ | $f_{Min}$ | $nbN$ | $Col$ | $LB$ | $CPU$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 16 | 5 | 16 | 24 | 652 | 13 | 61 |
| 2 | 5 | 15 | 4 | 19 | 12 | 398 | 14 | 47 |
| 3 | 4 | 22 | 3 | 24 | 19 | 751 | 18 | 72 |
| 4 | 5 | 20 | 5 | 20 | 22 | 784 | 17 | 86 |
| 5 | 4 | 88 | 3 | 88 | 5 | 302 | 85 | 49 |
| 6 | 3 | 86 | 3 | 86 | 14 | 356 | 80 | 186 |
| 7 | 4 | 121 | 3 | 128 | 8 | 3256 | 113 | 3362 |
| 8 | 15 | 146 | 12 | 158 | 24 | 9896 | 139 | 11986 |
| 9 | 16 | 141 | 13 | 156 | 17 | 1188 | 134 | 471 |
| 10 | 10 | 151 | 9 | 165 | 16 | 965 | 139 | 954 |

TABLE 7.4 – Résolution exacte du problème (*Branch and Bound* + génération de colonnes)

Il faut également remarquer que, pour la topologie en Ligne et sans fenêtre de temps (il s'agit donc du problème le plus simple), et malgré l'efficacité de l'algorithme, si l'on compare à la résolution directe du PLNE, il n'est pas envisageable de rechercher la solution optimale dès lors que l'on supervise un service réel. La résolution heuristique est donc importante et nous fournit, dès la 20ème réplication, des solutions toujours plus proches des 10% de l'optimale.

Dans le tableau 7.5, remarquons que $K_{min}$ et $K_{ins}$ obtiennent souvent des résultats très proches bien qu'un gap plus large commence à apparaître dans les instances de plus grande taille. Ceci est intéressant pour les cas réels, donc dans un contexte dynamique, où le nombre de véhicules déployés sera moins important que si $K_{opti}$ avait été à l'origine de la formation de la flotte.

| Inst. | $K_{ins,R=1}$ | $Gap_{f,R=1}$ | $CPU_{R=1}$ | $K_{ins,R=20}$ | $Gap_{f,R=20}$ |
|---|---|---|---|---|---|
| 1 | 5 | 6.2 | 0.1 | 5 | 0 |
| 2 | 4 | 6.6 | 0.2 | 4 | 6.6 |
| 3 | 3 | 9 | 0.1 | 3 | 4.5 |
| 4 | 5 | 15 | 0.2 | 5 | 5 |
| 5 | 4 | 9.1 | 0.2 | 4 | 6.8 |
| 6 | 3 | 8.1 | 0.3 | 3 | 4.6 |
| 7 | 3 | 6.6 | 0.4 | 4 | 4.1 |
| 8 | 12 | 8.2 | 0.6 | 13 | 3.4 |
| 9 | 14 | 14.1 | 0.3 | 15 | 8.4 |
| 10 | 11 | 13.2 | 0.5 | 10 | 7.9 |

TABLE 7.5 – Résolution approchée du problème (Insertions successives et courbes de Profil)

**DARPRel sur Circuit, avec $\delta = 0$.**

Le tableau 7.6 relève les résultats pour deux instances, l'une avec $|D| = 12$ et l'autre avec $|D| = 48$. On note $f_s$ le nombre d'arrêt associé à une solution réalisable créée aléatoirement et $f$ le nombre d'arrêt obtenu après optimisation. Ici, les heuristiques sont répliquées 1, 10, $10^2$ et $10^4$ fois ($R$).

| $R$ | $K$ | $S$ | $\|D\|$ | $f$ | $CPU$ | $f_s$ | $CPU_s$ |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 8 | 12 | 22 | 0 | 26 | 0 |
| 10 | 3 | 8 | 12 | 21 | 0 | 24 | 0 |
| 100 | 3 | 8 | 12 | 21 | 0,1 | 24 | 0,1 |
| 10000 | 3 | 8 | 12 | 21 | 9,2 | 23 | 12,9 |
| 1 | 5 | 8 | 48 | 59 | 0 | NA | NA |
| 10 | 5 | 8 | 48 | 59 | 0,2 | 74 | 0,1 |
| 100 | 5 | 8 | 48 | 59 | 1,3 | 74 | 1,4 |
| 10000 | 5 | 8 | 48 | 57 | 127,4 | 74 | 160,1 |

TABLE 7.6 – $f_s$ / $f$

Les deux résolutions indiquent un écart allant jusqu'à 17 arrêts. Si nous prenons la prochaine exploitation des VIPA sur le site de Ladoux de Michelin, le nombre d'arrêts sera aux alentours de 8 pour 2km de circuit. Si à un instant de la journée, 48 demandes sont émises, la résolution proposée dans ce chapitre peut donc engendrer 17 arrêts de véhicules en moins à partir d'une flotte de tournées vides par rapport à un management des requêtes "à la main". En considérant qu'un arrêt engendre une perte de 30 secondes (indépendamment du nombre de personnes entrantes et sortantes), un peu plus de 8 minutes seraient perdues et seulement 8 séries de 48

demandes seraient suffisantes pour que l'ensemble de la flotte perde, au final, plus d'une heure en raison d'une mauvaise orientation dans la construction des tournées. Enfin, ce rapide constat n'inclut même pas les problèmatiques de fiabilité, d'usure et d'énergie plus détaillées en début de chapitre.

Le tableau 7.7 relève les résultats obtenus au bout de 100 réplications (ce nombre semble être une bonne limite pour l'environnement d'exécution *mono-thread* utilisé pour ces tests pour les contraintes de réactivité d'un service de VIPA). Il différencie les deux amplitudes de fenêtres par les indices $_\theta$.

| $R$ | $K$ | $S$ | $\|D\|$ | $f_{\theta=5}$ | $CPU_{\theta=5}$ | $f_{\theta=10}$ | $CPU_{\theta=10}$ |
|---|---|---|---|---|---|---|---|
| 100 | 2 | 8 | 12 | 21 | 0,12 | 18 | 0,12 |
| 100 | 3 | 8 | 24 | 36 | 0,28 | 35 | 0,24 |
| 100 | 4 | 8 | 48 | 58 | 1,15 | 52 | 1,34 |

TABLE 7.7 – Résultats heuristiques sur différentes tailles de fenêtre de temps

Remarquons que, pour l'instance à 48 demandes, nous disposons d'un véhicule de moins que pour la même instance des expériences du tableau 7.6 et que, au bout de 10000 (ce qui n'est pas relevé ici) nous restons sur un nombre d'arrêts à 58 contrairement aux 57 obtenus avec plus de véhicules pour le premier tableau. Sans une totale certitude d'avoir obtenu un résultat optimal, nous pourrions avoir ici un cas où la résolution du problème *MinArret* engendrerait un nombre $K$ différent du minimum calculé par le problème *MinNbVehicules* comme c'est arrivé plusieurs fois dans la première série de tests.

Ensuite, il est clair que plus l'amplitude des fenêtres de temps est importante, plus il existe de solutions et, pour notre cas, plus le nombre d'arrêts est faible, sachant que les instances ne diffèrent que par l'amplitude des fenêtres des points *origine*.

Enfin, les temps obtenus pour la résolution de 100 réplications de notre heuristique restent particulièrement rapides et ceci grâce à l'heuristique contractant en amont les fenêtres de temps des nœuds de même label (même station). En effet, à peine plus d'une seconde est nécessaire pour 100 tentatives d'insertion des 48 demandes de la troisième instance (rappelons qu'ici le problème intégre les fenêtres de temps).

**DARPRel sur Circuit, avec $\delta$ non nul.**

Les tableaux 7.8 et 7.9 donnent les résultats sur les instances décrites dans le tableau 7.3. Le premier relève l'évolution des résultats selon le nombre de réplications $R$ de l'heuristique (de 1 à 1000) et le second, qui correspond aux même instances si ce n'est une durée maximale d'acheminement $\Delta^2$ plus grande, présente les résultats obtenus pour $R = 100$. Notons $f_{Opti}$ le nombre d'arrêts minimal obtenu pour une instance et un nombre de réplications $R$ donnés, $K_{Opti}$ est le nombre de véhicules nécessaires pour obtenir ce résultat. $f_{K_{Min}}$ et $K_{Min}$ ont les mêmes significations mais

| Inst.R | $f_{Opti}$ | $K_{Opti}$ | $f_{Min}$ | $K_{Min}$ | $T_{Global}$ | $T_{Ride}$ | $T_{Wait}$ | $CPU_R$ |
|---|---|---|---|---|---|---|---|---|
| $D60.1$ | 85 | 10 | 85 | 10 | 286.3 | 273.0 | 1.0 | 0.0 |
| $D60.10$ | 69 | 10 | 69 | 10 | 215.1 | 286.9 | 0.3 | 0.2 |
| $D60.10^2$ | 69 | 10 | 69 | 10 | 215.1 | 286.9 | 0.3 | 1.4 |
| $D60.10^3$ | 68 | 10 | 68 | 10 | 232.7 | 266.0 | 0 | 16.0 |
| $D90.1$ | 101 | 13 | 101 | 13 | 325.5 | 410.4 | 0 | 0.0 |
| $D90.10$ | 95 | 14 | 101 | 13 | 324.2 | 402.4 | 27.8 | 0.3 |
| $D90.10^2$ | 95 | 14 | 98 | 12 | 324.2 | 402.4 | 27.8 | 3.1 |
| $D90.10^3$ | 92 | 13 | 95 | 12 | 325.5 | 406.0 | 33.2 | 35.8 |
| $D120.1$ | 137 | 20 | 137 | 20 | 460.2 | 563.9 | 3.2 | 0.0 |
| $D120.10$ | 135 | 18 | 135 | 18 | 488.3 | 580.0 | 52.1 | 0.5 |
| $D120.10^2$ | 127 | 17 | 133 | 16 | 451.0 | 573.6 | 49.2 | 5.9 |
| $D120.10^3$ | 119 | 17 | 121 | 15 | 446.0 | 570.9 | 61.6 | 62.6 |
| $D150.1$ | 166 | 23 | 166 | 23 | 557.7 | 682.9 | 5.5 | 0.1 |
| $D150.10$ | 156 | 21 | 160 | 19 | 532.6 | 726.7 | 28.2 | 0.9 |
| $D150.10^2$ | 148 | 20 | 155 | 18 | 527.9 | 703.1 | 49.3 | 8.5 |
| $D150.10^3$ | 139 | 18 | 148 | 17 | 550.0 | 720.0 | 97.2 | 113.3 |
| $D180.1$ | 186 | 25 | 186 | 25 | 620.5 | 832.5 | 1.1 | 0.2 |
| $D180.10$ | 175 | 23 | 175 | 23 | 624.0 | 845.8 | 42.0 | 1.3 |
| $D180.10^2$ | 173 | 22 | 173 | 22 | 716.0 | 847.4 | 139.0 | 12.3 |
| $D180.10^3$ | 165 | 23 | 171 | 21 | 644.6 | 834.9 | 107.0 | 191.8 |
| $D216.1$ | 227 | 30 | 227 | 30 | 765.4 | 1023.4 | 6.5 | 0.2 |
| $D216.10$ | 214 | 27 | 214 | 27 | 776.0 | 1015.0 | 47.0 | 3.5 |
| $D216.10^2$ | 196 | 24 | 196 | 24 | 785.3 | 1035.2 | 161.2 | 37.8 |
| $D216.10^3$ | 187 | 24 | 187 | 24 | 751.3 | 1024.7 | 138.4 | 301.2 |
| $D252.1$ | 240 | 34 | 240 | 34 | 823.1 | 1192.8 | 15.4 | 0.3 |
| $D252.10$ | 225 | 30 | 225 | 30 | 856.7 | 1153.9 | 92.5 | 4.0 |
| $D252.10^2$ | 213 | 28 | 213 | 28 | 829.7 | 1221.0 | 145.6 | 38.4 |
| $D252.10^3$ | 213 | 28 | 224 | 28 | 829.7 | 1221.0 | 145.6 | 374.0 |
| $D288.1$ | 275 | 34 | 275 | 34 | 926.0 | 1363.0 | 3.7 | 0.4 |
| $D288.10$ | 252 | 33 | 252 | 33 | 931.3 | 1330.0 | 68.7 | 5.1 |
| $D288.10^2$ | 251 | 33 | 261 | 31 | 1004.9 | 1363.2 | 171.7 | 47.5 |
| $D288.10^3$ | 234 | 29 | 234 | 29 | 1037.2 | 1378.5 | 239.8 | 502.3 |

TABLE 7.8 – Résultats du DARPRel avec fenêtres de temps, temps de service et $\Delta^1$

cette fois-ci pour la réplication utilisant le plus petit nombre de véhicules ; en cas d'égalité, est choisie celle qui comporte le plus grand nombre d'arrêts ($f_{K_{Min}}$). Enfin, pour la réplication au plus petit nombre d'arrêts, nous notons :

— $T_{Global}$ : la somme des durées des tournées,
— $T_{Ride}$ : la somme des temps d'acheminement - ang. *Ride time* -,
— $T_{Wait}$ : la somme des temps d'attente aux stations avant service - ang. *Waiting time* -,
— $CPU$ : la durée en secondes de chaque exécution.

Les résultats montrent l'intérêt de la minimisation des arrêts sur les critères de performance introduits dans le chapitre 3.

| Inst.R | $f_{Opti}$ | $K_{Opti}$ | $f_{Min}$ | $K_{Min}$ | $T_{Global}$ | $T_{Ride}$ | $T_{Wait}$ | $CPU_R$ |
|---|---|---|---|---|---|---|---|---|
| $D60.10^2$ | 59 | 6 | 59 | 6 | 224.2 | 424.6 | 14.0 | 1.4 |
| $D90.10^2$ | 84 | 9 | 86 | 8 | 287.0 | 663.1 | 38.3 | 3.1 |
| $D120.10^2$ | 105 | 11 | 106 | 10 | 351.9 | 828.7 | 39.2 | 5.6 |
| $D150.10^2$ | 124 | 12 | 124 | 12 | 445.2 | 1097.8 | 85.2 | 9.0 |
| $D180.10^2$ | 145 | 14 | 145 | 14 | 533.7 | 1174.1 | 125.5 | 13.4 |
| $D216.10^2$ | 165 | 15 | 165 | 15 | 645.0 | 1449.0 | 175.8 | 20.1 |
| $D252.10^2$ | 194 | 19 | 194 | 19 | 570.9 | 1706.5 | 126.6 | 28.7 |
| $D288.10^2$ | 207 | 21 | 222 | 20 | 784.4 | 1948.5 | 156.8 | 39.0 |

TABLE 7.9 – Résultats du DARPRel avec fenêtres de temps, temps de service et $\Delta^2$

Sur les plus petites instances, les durées des tournées s'amenuisent au fil des réplications. Cependant, ce gain semble être de plus en plus absorbé par les temps d'attente (qui font partie du calcul des durées des tournées) si l'on analyse de plus grandes instances. Dans de futurs travaux, il serait intéressant d'entrer ces temps d'attente couplés aux nombres d'arrêts dans les critères de coûts.

Les résultats montrent que de contraintes larges sur les durées d'acheminement permet l'utilisation d'un nombre moins important de véhicules puisque ces derniers peuvent s'arrêter pratiquement à chaque station. Mais, évidemment, les usagers sont alors pénalisés et c'est d'autant plus vrai pour les trajets importants. Il semblerait qu'il soit intéressant de distinguer deux types de flottes : celles satisfaisant des demandes sur de petits trajets et d'autres sur de plus longs.

Enfin, nous remarquons sans surprise que, dans le tableau 7.9, les temps de connexion $T_{Ride}$ ont "explosé" du fait de $\Delta_2$. Le nombre de solutions étant alors plus important, ceci a permis de réduire le nombre d'arrêts.

## Perspectives et conclusion

Le problème de la minimisation des arrêts est un problème nouveau qui pourrait trouver sa place dans la littérature de la recherche opérationnelle. Outre le fait qu'elle correspond aux nouvelles flottes de véhicules autonomes, sa définition est simple : il s'agit de minimiser le nombre d'arrêts d'une flotte de véhicules satisfaisant des demandes le long d'un axe horizontal (ou d'une boucle), entrecoupé de stations. Ce problème a été mis en place suite à l'intégration d'une contrainte de fiabilité (qui subit la plus grande perte au moment de l'arrêt des véhicules) dans la perspective d'un nouveau type de transports à la demande basé sur les VIPA. Les *Private Rapid Transit* pourraient également en profiter, les sorties en stations étant, pour certaines topologies de circuits, particulièrement chronophages. Les possibilités envisageables d'applications de ces recherches sont nombreuses.

Ce chapitre, qui fait office d'introduction au problème, ouvre de nombreuses perspectives de travail. La première serait d'évaluer sa complexité (pour le cas en ligne dans un premier temps) ; les différents modèles présentés ici ainsi que les résultats obtenus lors des premières expérimentations nous amènent à penser que notre problème est NP-Complet lorsque les charges ne sont pas unitaires. Nous pouvons jusque là au moins noter que le problème est dans NP, la vérification de la faisabilité des solutions se faisant en temps polynomial.

Dans de futurs travaux, on pourra intégrer au problème les temps de service en les modélisant selon la somme de temps fixes et variables. Les temps fixes, correspondant aux opérations de court-circuitage, décélération, arrêts, ouverture/fermeture des portes, accélération et réintégration au sein du circuit, devront être sensiblement plus importants que les coûts variables qui ne modéliseront que le temps passé par les usagers (au nombre maximum de 6 pour les VIPA) pour sortir et entrer. En effet, même si le DARP, comme cela a été étudié en chapitre 3, pouvait les intégrer dans le calcul des instances, il faudrait faire ici une réelle étude de ces temps. L'algorithme général d'insertion aléatoire ne devra pas changer fondamentalement, les temps fixes devront juste être ajoutés, une seule fois pour chaque arrêt des véhicules, les temps variables s'additionnant à chaque demande traitée. Il sera intéressant de vérifier si la seule minimisation des durées des tournées implique une minimisation des arrêts et inversement.

Nous pouvons envisager de créer un système d'anticipation des demandes selon les bases expliquées dans le chapitre 6. Toujours à l'image de ce chapitre, nous pouvons également développer un ensemble de procédés sélectionnant et insérant une demande selon son impact (au niveau du nombre d'arrêts) sur les prochaines insertions des demandes restantes. Pour cela, imaginons que nous mettons en place, pour chaque tournée, une hiérarchie des demandes à insérer sur trois niveaux. Le premier concerne les demandes ayant ses deux arrêts en commun avec ceux de la tournée donnée, la seconde un seul arrêt identique et le dernier comprend l'ensemble des demandes restantes. L'insertion des demandes au sein d'une tournée peut modifier ces trois ensembles apparentés à la tournée modifiée. Il suffit alors d'analyser ces changements de "hiérarchie". La sélection de la demande et du véhicule sont alors

basés sur ces mouvements. L'impact d'une insertion en termes de nombre d'arrêts sur celui d'une prochaine demande à insérer peut s'évaluer par -2, -1, 0, 1 ou 2 arrêts. Ceci reprend le sens du calcul de l'*Insérabilité*.

Même si à première vue, et selon le degré d'abstraction utilisé par le présent modèle mais aussi la génération des instances, l'irrégularité de la localisation des stations sur le circuit paraît sans impact sur le service attendu, il faudra tout de même modéliser la construction des circuits pour optimiser le placement des stations selon les performances qui seront attendues. Ceci concerne à nouveau la recherche opérationnelle mais sort du contexte de la création de tournées de véhicules. Les AVG sont encore une bonne source de travaux de recherche pour ce type de problèmes.

Aussi, le passage au contexte dynamique devra pouvoir fournir rapidement une solution adaptable à l'exploitation d'un service tel que les VIPA. Ce passage sera relativement aisé en utilisant les techniques heuristiques présentées.

Enfin, ajoutons l'idée que la minimisation du nombre de fois où les véhicules s'arrêtent pourrait aider à nouveau à l'établissement des plannings des VIPA, si la localisation de ces arrêts était créée dynamiquement en fonction de celle des personnes désireuses d'être transportées. Situons-nous donc au moment où les personnes pourront se synchroniser facilement avec les véhicules partout où ils le désireront (par exemple avec les coordonnées GPS des deux agents) et lorsque ces engins motorisés pourront s'arrêter automatiquement sur la chaussée en tout point de la route sur laquelle ils évolueront. La création des stations se fera dynamiquement et l'intérêt de la minimisation des arrêts sera d'autant plus effective que ces stations dynamiques seront créées afin qu'elles soient en nombre limité tout en satisfaisant une contrainte de proximité avec les utilisateurs. Le dessin de gauche de la figure 7.5 schématise la création d'un seul arrêt "dynamique" pour trois usagers en minimisant la distance qu'ils ont à parcourir jusque là.

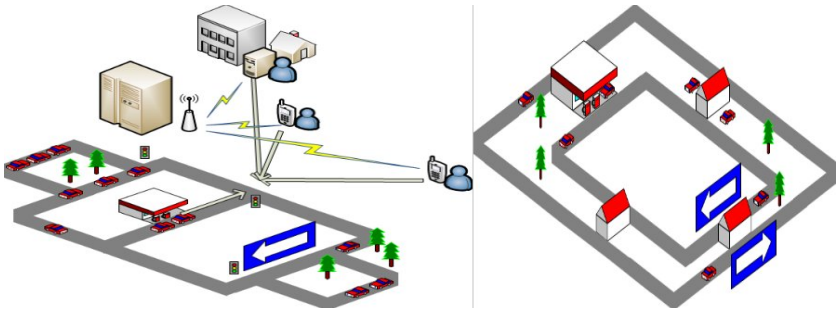

FIGURE 7.5 – Minimisation des arrêts avec stations dynamiques et sens bidirectionnel

Le dessin de droite pose le problème de la bidirectionnalité. Que les véhicules puissent fonctionner dans les deux sens ou bien que l'on dispose de deux flottes, l'attribution des véhicules aux usagers doit également être optimisée et ceci en intégrant toujours la minimisation des arrêts.

# Chapitre 8

# Heuristique lagrangienne pour un problème de *Lot-Sizing* avec capacités de transfert et stockage

## Introduction et état de l'art

La compétitivité toujours croissante des entreprises a rapidement amené ces dernières à limiter les coûts à travers toutes leurs chaînes logistiques, de l'approvisionnement en matières premières à la distribution. Il faut aussi considérer, dans le même ordre d'idées, la production mais aussi l'acheminement des produits à la fois dans l'espace et dans le temps. Elmaghraby (1978) nous rappelle que l'étude de la planification de la production a commencé au début du XX$^{\text{ième}}$ siècle avec l'*Economic Lot and Scheduling Problem* (ELSP). Par la suite, les besoins du marché (la demande), évoluant au cours du temps pour la plupart des manufactures, ont impliqué un élargissement de l'ELSP et plus précisément les premiers grands travaux sur les problèmes de *Lot-Sizing* (LSP) réalisés par Wagner and Whitin (1958). Depuis, les LSP continuent à être étudiés dans les laboratoires et les solutions sont exploitées dans les usines.

Une des principales difficultés des LSP découle de la restriction des capacités de production dans chaque période de temps. Sans capacité, le problème se résout polynomialement tout comme le problème avec capacités de production constantes pour chaque période (Florian and Klein (1971)). Lorsque ces capacités varient sur l'horizon de planification, le problème devient NP-Difficile (Chen and Thizy (1990)).

Les problèmes réels de *Lot-Sizing* sont généralement contraints par des capacités de production, par d'importants coûts fixes et par la spécificité de chaque environnement industriel. Le problème standard multi-produits avec capacité - ang. *Multi-item Capacitated Lot-Sizing problem* (MCLS) - est le problème qui a été le plus largement étudié (Maes and Van Wassenhove (1988)). Cependant, pour le MCLS, obtenir la solution optimale et même, parfois, une qui soit juste réalisable, reste compliqué. Chen and Thizy (1990) prouvent que le MCLS est NP-Difficile au sens fort - ang. *strongly NP-Hard* -. Florian et al. (1980) et Drexl and Kimms (1997) montrent que

le LSP à un seul type de produit est NP-Difficile au sens faible - ang. *ordinary NP-Hard* -. Le LSP avec capacité est donc particulièrement difficile à résoudre, d'autant plus si l'on considère les différentes extensions dues au contexte industriel.

Il existe un nombre considérable de ramifications des problèmes de *Lot-Sizing* (cf. Lawler et al. (1993), Pochet and Wolsey (2006) et Potts and Wassenhove (1992)) comme l'intégration de capacités de stockage et de transfert, l'ajout de coûts et temps fixes - ang. *setup costs/times* - (Trigeiro et al. (1989)), de fenêtres de temps ou encore du report de la demande (Kedad-Sidhoum et al. (2009)). Les modèles peuvent être dynamiques ou statiques et peuvent même intégrer un processus d'assemblage, sur une ou plusieurs usines et pour un ou plusieurs produits. Les critères de performance sont souvent la minimisation des coûts de production, du coût des retards et des coûts liés à la non production de certains produits (pénalités). Jans and Degraeve (2008a), Lawler et al. (1993) et Graves et al. (1993) réalisent trois vastes états des connaissances sur le sujet et Jans and Degraeve (2008b) proposent une large présentation des derniers modèles développés dans le domaine du *Lot-Sizing*.

La diversité ne s'arrête pas là, elle est également à l'œuvre dans les différentes modélisations de chaque problème ainsi que dans les techniques de résolution (relaxation lagrangienne (Sambasivan and Yahya (2005) et Chen and Thizy (1990)), programmation dynamique (Aksen et al. (2003)), etc.).

Le problème traité dans ce chapitre est multi-usines, multi-produits et multi-périodes. Il considère des capacités à la production, sur les transferts et sur le stockage de produits. Il intègre aussi des coûts et des temps fixes à la production. Le problème est statique, dans le sens où toute la demande est connue à l'avance. Celle-ci est différente selon les régions, selon les produits et selon la période. Les produits peuvent être directement fournis aux clients d'une usine (une usine pour une région). Une usine peut transférer des produits à une autre usine et/ou les stocker. Enfin, le planning de production calculé doit minimiser les coûts globaux de production (fixes et variables), de stockage, et de transfert.

Ce chapitre résout des instances de ce problème, et expose comment elles peuvent être décomposées, à l'aide d'un schéma lagrangien, en un problème maitre de localisation et un problème esclave de multi-flots à coût minimum. La décomposition lagrangienne du problème maître part d'une adaptation des travaux de Sambasivan and Yahya (2005) qui ne considèrent pas de capacité de stockage et de transfert. Au terme de ce chapitre, le modèle mathématique du problème général permet de comparer les résultats obtenus avec ceux des heuristiques de la décomposition.

## 8.1 Définition du problème et formulation mathématique

On considère un ensemble $\mathcal{K}$ de $K$ objets (produits) qui doivent être fabriqués par $I$ usines sur un horizon de temps discrétisé de $T$ périodes. On note $\mathcal{I}$ l'ensemble des usines et $\mathcal{T}$ celui des périodes. Une usine $i$ dans $\mathcal{I}$ peut-être à la fois une usine productrice, lorsque des objets sont produits en $i$, et une usine fournisseur si $i$ doit répondre à une demande. Une usine $i$ peut être aussi utilisée comme un espace de stockage ou encore comme l'émetteur, le relais ou le récepteur de produits faisant l'objet de transfert.

Nous devons satisfaire une demande $d_{it}^k$ pour chaque période $t$ dans $\mathcal{T}$, avec $k$ un type de produit de $\mathcal{K}$, et une usine $i$ de $\mathcal{I}$. On suppose que la production et les transferts sont effectués dans une même période de temps. En effet, si un objet est fabriqué à l'usine $i$ à la période $t$ et est transféré à l'usine $j$, $j \neq i$, le transfert a lieu à la période $t$. Par contre, si un objet est stocké dans l'usine $j$, on considère que le stockage a lieu entre la période $t$ et la période $t+1$.

La production d'un objet $k$ à l'usine $i$ et à la période $t$ induit un coût variable de production $p_{it}^k$, le coût fixe étant $q_{it}^k$. Le coût de stockage $h_{it}^k$ intervient pour chaque unité d'un produit $k$ stockée en usine $i$ à la période $t$. $c_{ijt}^k$ est le coût de transfert du produit $k$ de l'usine $i$ à l'usine $j$ en période $t$. L'objectif est de minimiser le coût total, i.e. coûts fixes et variables de production, et coûts de transfert/stockage, pour satisfaire la demande sur les $T$ périodes.

Les contraintes de capacité sont considérées à la fois sur la production, le transfert et le stockage. On note $\alpha_{it}^k$ la quantité de capacité de production consommée par unité fabriquée du produit $k$ par l'usine $i$ à la période $t$. De la même manière, $\beta_{it}^k$ correspond à la capacité fixe consommée par l'ouverture de l'usine $i$ pour fabriquer l'objet $k$ à la période $t$. Pour chaque unité du produit $k$, $\gamma_{it}^k$ est la quantité de capacité de stockage consommée par l'usine $i$ à la période $t$. On considère que la quantité de capacité de production (resp. de stockage) disponible à l'usine $i$ en période $t$ est $A_{it}$ (resp. $B_{it}$). Les opérations de transfert sont également contraintes à de telles capacités. Chaque unité de produit $k$ transférée de l'usine $i$ à l'usine $j$ à la période $t$ consomme $\tau_{ijt}^k$. La consommation totale induite par le transfert d'objets de $i$ vers $j$ en $t$ ne doit pas excéder $BT_{ijt}$.

Le problème consiste à planifier les opérations de production et les opérations de transfert/stockage pour satisfaire la demande et de façon à minimiser la somme des coûts.

Nous proposons un modèle mathématique mixte en nombres entiers. Les variables de décision sont définies, pour tout $i, j$ dans $\mathcal{I}$, $k$ dans $\mathcal{K}$, et $t$ dans $\mathcal{T}$, de la manière suivante :

- $x_{it}^k$ représente la quantité (positive) d'objets $k$ produite par l'usine $i$ à la période $t$,
- $z_{it}^k$ est une variable binaire égale à 1 si l'objet $k$ est produit par l'usine $i$ à la période $t$, 0 sinon,
- $s_{it}^k$ représente la quantité (positive) d'objets $k$ stockée par l'usine $i$ à la fin de la période $t$ (on suppose que $s_{i0}^k = 0$),
- $y_{ijt}^k$ donne la quantité (positive) d'objets $k$ transférée de l'usine $i$ à l'usine $j$ à la période $t$.

Le problème de *Lot-Sizing* multi-produits, multi-usines avec capacités de production, stockage et transfert - ang. *multi-item Multi-plant Lot-sizing problem under Storage, Transfer and Production Capacity constraints* (MLS-STPC) - peut être formulé comme suit :

$$\min \quad \sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}}(p_{it}^k x_{it}^k + q_{it}^k z_{it}^k + h_{it}^k s_{it}^k) + \sum_{i,j\in\mathcal{I}}\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}} c_{ijt}^k y_{ijt}^k \tag{8.1}$$

s.t.

$$x_{it}^k + s_{it-1}^k + \sum_{j\in\mathcal{I}} y_{jit}^k = d_{it}^k + s_{it}^k + \sum_{j\in\mathcal{I}} y_{ijt}^k, \quad \forall i,k,t \tag{8.2}$$

$$\sum_{k\in\mathcal{K}} \alpha_{it}^k x_{it}^k + \beta_{it}^k z_{it}^k \leq A_{it}, \quad \forall i,t \tag{8.3}$$

$$\sum_{k\in\mathcal{K}} \gamma_{it}^k s_{it}^k \leq B_{it}, \quad \forall i,t \tag{8.4}$$

$$\sum_{k\in\mathcal{K},i\in\mathcal{I}} \tau_{ijt}^k y_{ijt}^k \leq BT_{ijt}, \quad \forall j,t \tag{8.5}$$

$$x_{it}^k \leq \sum_{l=1}^{N}\sum_{r=t}^{T} d_{lr}^k z_{it}^k, \quad \forall i,k,t \tag{8.6}$$

$$x_{it}^k, s_{it}^k, y_{ijt}^k \geq 0, \quad \forall i,j,k,t \tag{8.7}$$

$$z_{it}^k \in \{0,1\}, \quad \forall i,k,t \tag{8.8}$$

La fonction objective (8.1) minimise les coûts totaux de production (fixes et variables), de transfert et de stockage. Le solde d'inventaire de la quantité de produits $k$ durant la période $t$ dans l'usine $j$ est donné par la contrainte (8.2). La contrainte de capacité de production (resp. stockage) à l'usine $i$ en période $t$, est donnée par (8.3) (resp. (8.4)).

De la même manière, la contrainte de capacité sur les opérations de transfert de produits de l'usine $i$ vers l'usine $j$ à la période $t$ est donnée par (8.5). Enfin, la contrainte (8.6) est dite liante.

## 8.2 Reformulation du MLS-STPC en problème de multi-flots

Le problème MLS-STPC peut être formulé comme un problème de multi-flots - ang. *multicommodity flow problems* - avec coûts fixes. Plusieurs travaux de la littérature traite de la formulation du problème multi-produits et avec capacité de production : relevons Barany et al. (1984) qui proposent une méthode exacte basée sur une formulation multi-flots pour le problème sans coût fixe. On définit une reformulation du problème MLS-STPC basée sur un réseau multi-flots. Cette reformulation est particulièrement utile dans les sections suivantes pour décrire notre schéma de décomposition.

On considère un réseau $G = (V, A)$ associé au problème MLS-STPC de telle sorte que l'ensemble de nœuds soit défini par $V = \{(i,t), i \in \mathcal{I}, t \in \mathcal{T}\} \cup \{T+1, 0\}$ où 0 et $T+1$ sont deux nœuds virtuels qui représentent respectivement la source et le puits. Les arcs de A font partie d'un des ensembles suivants :

- l'ensemble $A_s$ d'arcs de stockage $((i,t),(i,t+1))$ qui définit l'inventaire total de l'usine $i$ à la fin de la période $t$, pour $i = 1, \ldots, N,\ t = 1, \ldots, T-1$,
- l'ensemble $A_t$ d'arcs de transfert défini par $((i,t),(j,t)), t \in \mathcal{T}, i, j \in \mathcal{I}, i \neq j$, qui représente les quantités transférées de l'usine $i$ à l'usine $j$,
- l'ensemble $A_p$ d'arcs de production défini par $(0,(i,t)), i \in \mathcal{I}, t \in \mathcal{T}$, représentant les quantités produites à l'usine $i$ à la période $t$,
- l'ensemble $A_c$ d'arcs de demande $((i,t), T+1), i \in \mathcal{I}, t \in \mathcal{T}$, qui définit la quantité consommée à l'usine $i$ en période $t$,
- l'arc additionnel $(T+1, 0)$ appelé arc d'équilibre - ang. *equilibrium arc* -.

L'ensemble des arcs $A$ du graphe $G$ est par conséquent défini de la manière suivante :

$$A = A_s \cup A_t \cup A_p \cup A_c \cup \{(T+1, 0)\}$$

Chaque nœud $a$ de $A$ est caractérisé par un flot, un coût et une capacité. Deux éléments de coûts sont définis, pour chaque arc $a$ : un coût fixe $FC^k(a)$ et un coût variable $VC^k(a)$ pour tout produit $k \in \mathcal{K}$, comme suit :

- si $a = ((i,t),(i,t+1))$ est un arc de stockage, alors $FC^k(a) = 0$ et $VC^k(a) = h^k_{it}$,
- si $a = ((i,t),(j,t))$ est un arc de transfert, alors $FC^k(a) = 0$ et $VC^k(a) = c^k_{ijt}$,
- si $a = (0,(i,t))$ est un arc de production, alors $FC^k(a) = q^k_{it}$ et $VC^k(a) = p^k_{it}$,
- si $a = ((i,t), T+1))$ est un arc de demande, alors $FC^k(a) = 0$ et $VC^k(a) = 0$,
- si $a = (T+1, 0)$, alors $FC^k(a) = 0$ et $VC^k(a) = 0$.

De même, on définit les consommations de capacité (fixe : $FCP^k(a)$ ; variable : $VCP^k(a)$), pour tout arc $a$ de $A$ et pour chaque produit $k$, comme suit :

- si $a = ((i,t),(i,t+1))$ est un arc de stockage, alors $FCP^k(a) = 0$ et $VCP^k(a) = \gamma^k_{it}$,
- si $a = ((i,t),(j,t))$ est un arc de transfert, alors $FCP^k(a) = 0$ et $VCP^k(a) = \tau^k_{ijt}$,
- si $a = (0,(i,t))$ est un arc de production, alors $FCP^k(a) = \beta^k_{it}$ et $VCP^k(a) = \alpha^k_{it}$,

— si $a = ((i,t), T+1)$ est un arc de demande, alors $FCP^k(a) = 0$ et $VCP^k(a) = 0$,
— si $a = (T+1, 0)$, alors $FCP^k(a) = 0$ et $VCP^k(a) = 0$.

La capacité $CA(a)$ d'un arc $a$ peut être définie de la manière suivante :

— si $a = ((i,t),(i,t+1))$ est un arc de stockage, alors $CA(a) = B_{it}$,
— si $a = ((i,t),(j,t))$ est un arc de transfert, alors $CA(a) = BT_{ijt}$,
— si $a = (0,(i,t))$ est un arc de production, alors $CA(a) = A_{it}$,
— si $a = ((i,t), T+1)$ est un arc de demande, alors $CA(a) = +\infty$,
— si $a = (T+1, 0)$, alors $CA(a) = +\infty$.

On dénote par la suite $d^k_{it}$ de $D^k(a)$ si $a = ((i,t), T+1)$, i.e. $a$ est un arc de demande.

La figure 8.1 illustre une instance du problème pour un unique produit $k$, deux usines $i$ et $i+1$ et deux périodes $t$ et $t+1$. Les valeurs entre parenthèses représentent la demande alors que les autres sont des coûts.

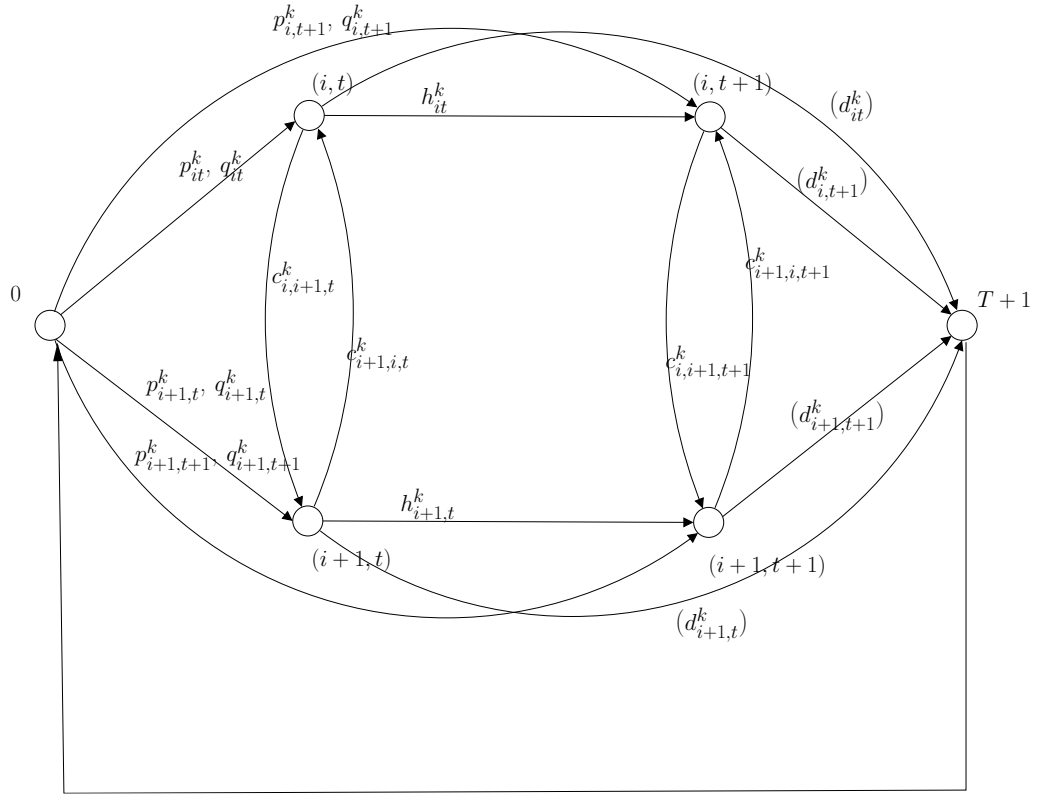


FIGURE 8.1 – Exemple du problème MLS-STPC avec 2 usines et 2 périodes

On note $f$ le multi-flots et $Ind(f)$ le vecteur de booléens indexé sur les arcs avec, pour tout $a \in A$, $Ind(f(a)) = 1$ si $f(a) > 0$ et 0 sinon. De plus, on note $(a^-, a^+)$ chaque arc $a$, où $a^-$ et $a^+$ représentent les extrémités (*endvertices*), et par $A_{pst}$ (resp. $A_{st}$) l'ensemble $A_p \cup A_s \cup A_t$ (resp. $A_s \cup A_t$).

La reformulation du problème MLS-STPC est la suivante :

$$\min \sum_{k\in\mathcal{K},a\in A_p} FC^k(a)Ind(f^k(a)) + \sum_{k\in\mathcal{K},a\in A_{pst}} VC^k(a)f^k(a) \tag{8.9}$$

$$\sum_{k\in\mathcal{K}}(VCP^k(a)f^k(a) + FCP^k(a)Ind(f^k(a))) \leq CA(a) \quad \forall a \in A_{pst} \tag{8.10}$$

$$f^k(a) = D^k(a) \quad \forall a \in A_c,\ k \in \mathcal{K} \tag{8.11}$$

$$\sum_b f^k((b,a^-)) = \sum_b f^k((a^+,b)) \quad \forall a \in A,\ k \in \mathcal{K} \tag{8.12}$$

Les contraintes (8.10) sont les contraintes de capacité. Satisfaire la demande pour chaque produit est imposée par la contrainte (8.11). Enfin, la contrainte (8.12) est celle de la conservation du flot. Il découle de cette formulation que résoudre le problème MLS-STPC revient à rechercher, sur le graphe $G$, le multi-flots $f$ qui minimise le coût total.

## 8.3 Résolution du MLS-STPC par relaxation lagrangienne

Comme mentionné en section 8, le problème MLS-STPC est NP-Difficile. On propose, pour le résoudre, au travers d'un schéma lagrangien, une décomposition du problème en un problème de localisation et de multi-flots coût minimum. L'heuristique lagrangienne est basée sur la relaxation des contraintes de capacité, c'est-à-dire les contraintes (8.10) dans la formulation mixte en nombres entiers de la section 8.2. Le vecteur de multiplicateurs de Lagrange est un vecteur $\lambda$ indexé sur l'ensemble $A_{pst}$ de valeurs positives. La fonction de Lagrange $L(f,\lambda)$ est alors définie comme suit :

$$\begin{aligned}
&L(f,\lambda) = \\
&\textstyle\sum_{k\in\mathcal{K},a\in A_p} FC^k(a)Ind(f^k(a)) + \sum_{k\in\mathcal{K},a\in A_{pst}} VC^k(a)f^k(a) \\
&\textstyle+ \sum_{a\in A_{pst}} \lambda(a) \sum_k FCP^k(a)Ind(f^k(a)) \\
&\textstyle+ \sum_{a\in A_{pst}} \lambda(a) \sum_k VCP^k(a)f^k(a) - \sum_{a\in A_{pst}} \lambda(a)CA(a)
\end{aligned}$$

Ainsi, le problème lagrangien est : $\min_{\lambda,f} L(f,\lambda)$ tel que les contraintes (8.11) et (8.12) sont satisfaites.

Soit un vecteur de multiplicateurs de Lagrange $\lambda = (\lambda(a), a \in A_{pst}) \geq 0$, ainsi, pour toute paire de nœuds $(v,w)$ dans $V\backslash\{0,T+1\}$, nous fixons :

- $P^k_\lambda(v) = FC^k((0,v)) + \lambda((0,v))FCP^k((0,v))$,
- $Q^k_\lambda(v,w) = VC^k((0,v)) + \lambda((0,v))VCP^k(0,v) + Dist^k_\lambda(v,w))d^k_{w,T+1}$, où $Dist^k_\lambda$ $(v,\ w)$ est la longueur du plus court chemin de $v$ à $w$ dans le réseau $G$, calculée en considérant tout arc $a \in A_{st}$ de longueur $VC^k(a) + \lambda(a)VCP^k(a)$.

Rappelons brièvement la définition du problème (sans capacité) de localisation utile à la résolution du problème MLS-STPC (cf. Mirchandani and Francis (1990)). Soit un ensemble de nœuds $X$, une fonction $g$ allant de $X$ à l'ensemble des rationnels positifs, et une fonction $l$ allant de $X \times X$ à l'ensemble des rationnels positifs,

telles que $l(x,x) = 0$ pour tout $x$ dans $X$. Le problème de localisation consiste à rechercher un sous-ensemble $Y$ de $X$ qui minimise la quantité : $\sum_{x\in Y} g(x) + \sum_{x\in X} \min_{y\in Y} l(x,y)$.

On a le théorème suivant :

**Théorème 1** : Minimiser $L(f,\lambda)$, sur l'ensemble des multi-flots $f$ qui satisfait la demande de la contrainte (8.11), revient à résoudre, pour tout $k$ de $K$, une instance d'un problème de localisation notée $FL^k_\lambda$ sur $X = V\backslash\{0, T+1\}$ avec $g = P^k_\lambda$ et $l = Q^k_\lambda$. La valeur optimale associée $\min_f L(f,\lambda)$ est alors égale à $W^k_\lambda - \sum_{a\in A_{pst}} \lambda(a)CA(a)$ où $W^k_\lambda$ est la valeur optimale de l'instance du problème de localisation $FL^k_\lambda$.

**Preuve.** Résoudre de manière optimale une instance de $FL^k_\lambda$ fournit un ensemble de nœuds $Y^*$ qui minimise $\sum_v P^k_\lambda(v) + \sum_v \min_w Q^k_\lambda(v,w)$. De plus, chaque nœud $v \in Y^*$ sera associé à un arc de production $(0,v)$, et un arc de la demande $(w, T+1)$ sera satisfait en prenant le plus court chemin entre $v$ et $w$, c'est à dire celui qui minimise le coût total $VC^k(a) + \lambda(a)VCP^k(a)$ sur l'ensemble des arcs du chemin. De plus, la valeur $\sum_{v\in Y} P^k_\lambda(v)$ est minimisée, ce qui correspond à la minimisation du coût total des coûts fixes de la fonction lagrangienne, c'est-à-dire $FC^k((0,v)) + \lambda((0,v))FCP^k((0,v))$. Si nous fixons tous les producteurs actifs comme un ensemble d'arcs $(0,v)$ tel que $v \in Y^*$, alors chaque nœud de la demande $w$ obtient la totalité de la demande $d^k_{w,T+1}$ du nœud $v$ dans $Y^*$ qui induit le meilleur coût. **Fin Preuve.**

Nous décrivons ensuite l'heuristique lagrangienne résolvant le problème MLS-STPC. Dans un premier temps, cette heuristique résout une instance du problème de localisation. Nous supposons qu'une méthode FACLOC, qui utilise le paramètre $N$ pour le nombre de réplications, est exécutée pour résoudre des instances du problème de localisation. FACLOC est spécifiée en section 8.4.2 où deux alternatives sont considérées. La première consistera en l'utilisation du *solver* CPLEX. La seconde sera basée sur une heuristique.

Calculer la valeur maximale $\max_{\lambda\geq 0} \min_f L(f,\lambda)$ peut se faire par la procédure LAG-LOT-SIZING suivante :

**Algorithme 30** LAG-LOT-SIZING

1: $\lambda \leftarrow 0$ ; $W \leftarrow 0$ ; $\lambda_{max} \leftarrow 0$ ; $n \leftarrow 0$ ;
2: **Tant que** $W$ est amélioré en moins de N itérations **Faire**
3: calculer $W_\lambda$ en appliquant la procédure FACLOC à $FL^k_\lambda$ ;
4: Si $W_\lambda > W$ Alors $W \leftarrow W_\lambda$ ; $\lambda_{max} \leftarrow \lambda$ ; mettre à jour $A^\lambda_p$ ;
5: mettre à jour $\lambda$ de façon à ce que $(\lambda_n)$ satisfasse les conditions de convergence du procédé de sous-gradient ($n \to +\infty$, $\lambda_n \to 0$ et $\sum_i \lambda_i \to +\infty$) ;
6: mettre à jour la matrice des distances $Dist^k_\lambda$ ; $n \leftarrow n+1$ ;
7: **Fin tant que**

L'ensemble $A_p^\lambda$ correspond aux arcs actifs de production liés à la solution finale. Plusieurs formules peuvent être utilisées pour calculer la séquence des $\lambda_n$. Elles permettent une mise à jour efficace des multiplicateurs de Lagrange (cf. Minoux (1983)).

LAG-LOT-SIZING se termine en donnant une valeur $W$, un vecteur de multiplicateurs $\lambda_{max}$ et un sous-ensemble $A_p^\lambda$ d'arcs de production actifs, c'est-à-dire des arcs avec un flot strictement positif. Cependant, les contraintes de capacité étant relaxées, la faisabilité de la solution n'est pas garantie. C'est pourquoi une heuristique appelée PROJECTION, qui recherche une solution réalisable, est proposée ensuite.

PROJECTION est basée sur la résolution d'une instance du problème MLS-STPC relaxé et sur une solution obtenue par l'algorithme LAG-LOT-SIZING. Les étapes principales de l'algorithme PROJECTION sont les suivantes :

1. Résoudre le problème de multi-flots de coût minimal pour $G' = (V', A')$ où $A' = A_p^\lambda \cup A_{stc} \cup \{(T+1, 0)\}$ et $S'$ est le sous-ensemble $S$ induit par $A'$ et où tous les paramètres fixes sont fixés à 0. La solution obtenue de cette étape est notée $\bar{f}^k(a)$ ;
2. Résoudre l'instance suivante du problème MLS-STPC en utilisant l'algorithme LAG-LOT-SIZING pour $G = (V, A)$ :
   — pour tout arc $a \in A_c$, $D^k(a) \leftarrow D^k(a) - \bar{f}^k(a)$, (demande résiduelle)
   — pour tout arc $a \in A_p^\lambda$, $CA(a) \leftarrow CA(a) - \sum_{k\in\mathcal{K}} VCP^k(a)\bar{f}^k(a) - \sum_{k\in\mathcal{K}} FCP^k(a)$, (capacité résiduelle)
   — pour tout arc $a \in A_{st}$, $CA(a) \leftarrow CA(a) - \sum_{k\in\mathcal{K}} VCP^k(a)\bar{f}^k(a)$. (capacité résiduelle)

La première étape de l'algorithme PROJECTION consiste en la résolution du problème classique de multi-flots de coût minimal avec un *solver* standard. La seconde étape de PROJECTION fournit un ensemble de producteurs actifs noté $\bar{A}_p^\lambda$. PROJECTION ne fournissant pas nécessairement une solution réalisable, nous proposons une procédure itérative MLS-STPC-ALG afin d'en rechercher une réalisable, c'est à dire où toute la demande est satisfaite en respectant toutes les restrictions de capacité. Cette méthode est décrite page suivante.

**Algorithme 31** MLS-STPC-ALG

Stop $\leftarrow 0$ ; $A_p^\lambda, \bar{A}_p^\lambda \leftarrow \emptyset$ ;
initialiser N ; initialiser une instance $I_G$ de MLS-STPC avec les paramètres d'origine ;
**Tant que** Stop $= 0$ **Faire**
  appliquer LAG-LOT-SIZING(N) sur l'instance $I_G$ ; mettre à jour $A_p^\lambda$ ;
  **Si** $A_p^\lambda = \emptyset$ **Alors**
    Stop $\leftarrow 0$ (solution non réalisable) ;
  **Sinon**
    calculer $\bar{f}$ avec PROJECTION étant donné $A_p^\lambda$ ;
    **Si** $\bar{f}$ est nul **Alors**
      Stop $\leftarrow 0$ (solution non réalisable) ;
    **Sinon**
      mettre à jour $I_G$ avec la demande et la capacité résiduelle ;
      Si $D^k(a) = 0$ pour tout $a \in A_c$ Alors Stop $\leftarrow 0$ (solution réalisable) ;
    **Fin si**
  **Fin si**
**Fin tant que**

La procédure MLS-STPC-ALG fournit au moins une borne MIN sur la fonction Objectif du problème MLS-STPC. Quand les contraintes sont satisfaites, une solution réalisable est obtenue.

## 8.4 Résultats expérimentaux

Les instances du problème de localisation ont été alternativement résolues par CPLEX et l'heuristique FACLOC, les deux s'exécutant sur un AMD opteron 2.1GHz. Nous testons plusieurs instances basées sur celles décrites par Sambasivan and Yahya (2005). Le générateur d'instances correspond aux paramètres suivants, avec UD signifiant "Distribution Uniforme", :

— nombre d'usines : $I = 2, 3, 4, 5, 8, 10$,
— nombre d'objets : $K = 2, 3, 5, 10, 15$,
— nombre de périodes : $T = 3, 5, 6, 10, 15$,
— coût variable de production donné par UD[0,8],
— coût variable de stockage donné par UD[0, 2],
— coût variable de transfert donné par UD[0, 2],
— coût fixe (*setup*) de production : UD[5, 15],
— demande donnée par UD[0, 50].

La taille d'une instance est définie par le triplet $(I, K, T)$. Les capacités ont été générées comme suit : un premier flot $f$ a été créé "aléatoirement", de telle sorte qu'il satisfasse l'ensemble de la demande. Une valeur $\alpha$ est donnée par UD[0, 100], et les capacités des différents arcs ont été calculées de façon à rendre possible une augmentation de $f$ sur l'arc d'équilibre par $\alpha\%$.

On note $R_{CP}(k)$ le ratio moyen sur la base "capacité/demande" utilisé pour interpréter la difficulté des contraintes sur la capacité de production d'un objet $k$. Le même ratio est utilisé pour l'ensemble des instances mais la distribution change pour chaque couple $(i, t)$. On génère des instances avec $R_{CP}(k) = 30\%, k \in K$, pour les expérimentations des sections 8.4.1 et 8.4.2. Les instances générées en section 8.4.3 sont telles que $R_{CP}(k) = 0$.

Nous résolvons les instances du problème de localisation de manière exacte, en utilisant CPLEX et nous obtenons le tableau 8.1 qui relève les gaps obtenus en pourcentage. Les notations suivantes sont utilisées :

— Ite : nombre d'itérations de la procédure MLS-STPC-ALG,
— Gap : donné par $\frac{W-W_{Opt}}{W_{Opt}} * 100$ qui est le gap entre la valeur $W$ donnée par la procédure MLS-STPC-ALG et la valeur optimale $W_{Opt}$ obtenue par CPLEX,
— GLg : donné par $\frac{W-W_{Lag}}{W_{Lag}} * 100$ qui est le gap entre la valeur $W$ et la valeur $W_{Lag}$ fournie par l'algorithme LAG-LOT-SIZING,
— GLP : donné par $\frac{W-W_{LP}}{W_{LP}} * 100$ qui est le gap entre la valeur $W$ et la valeur $W_{LP}$ produite par la relaxation fractionnaire du problème MLS-STPC,
— GPr : donné par $\frac{W_{opt}-W_{opt1}}{W_{opt1}} * 100$ qui est le gap entre la valeur $W_{opt}$ et la valeur optimale $W_{opt1}$ du problème ne prenant en compte que la capacité de production,
— GFr : donné par $\frac{W_{opt}-W_{opt\cdot free}}{W_{opt\cdot free}} * 100$ qui est le gap entre la valeur $W_{opt}$ et la valeur optimale $W_{opt\cdot free}$ du problème sans aucune capacité,
— Def : pourcentage de la demande non satisfaite après la première itération de l'algorithme MLS-STPC-ALG.

### 8.4.1 Expérimentations avec résolution exacte du problème de localisation - partie 1

Le tableau 8.1 relève les résultats obtenus en résolvant le problème de localisation de manière exacte avec CPLEX.

| Size | Ite | Gap (%) | GLg (%) | GLP (%) | GPr (%) | GFr(%) | Def (%) |
|---|---|---|---|---|---|---|---|
| (3,5,3) | 1 | 0 | 5.4 | 15.6 | 0.8 | 85 | 0 |
| (3,5,3) | 1 | 0 | 7.5 | 12.7 | 0.6 | 54 | 0 |
| (3,5,3) | 1 | 0 | 0.8 | 15.9 | 5.8 | 32.5 | 0 |
| (4,10,6) | 1 | 1.7 | 0.2 | 15.6 | 2.9 | 21.5 | 0 |
| (4,10,6) | 1 | 0 | 0.2 | 20.3 | 3.8 | 9.2 | 0 |
| (4,10,6) | 2 | 1.0 | 5.4 | 13.8 | 20.0 | 25.4 | 3.5 |
| (5,10,10) | 1 | 0.7 | 0.6 | 19.7 | 5.2 | 20.2 | 0 |
| (5,10,10) | 1 | 0.7 | 1.5 | 15.3 | 6.7 | 19.1 | 0 |
| (5,10,10) | 1 | 0.1 | 0.4 | 20.2 | 5.5 | 14.7 | 0 |
| (8,10,15) | 1 | 0 | 0.9 | 13.5 | 2.6 | 23.6 | 0 |
| (8,10,15) | 1 | 0.8 | 3.4 | 12.8 | 3.7 | 49.0 | 0 |
| (8,10,15) | 2 | 1.5 | 4.5 | 19.4 | 7.5 | 58.3 | 2.2 |
| (10,15,15) | 1 | 1.2 | 2.6 | 23.7 | 3.4 | 37.2 | 0 |
| (10,15,15) | 1 | 0 | 0.9 | 14.5 | 3.5 | 29.6 | 0 |
| (10,15,15) | 3 | 2.5 | 6.5 | 28.2 | 11.8 | 82.0 | 4.8 |

TABLE 8.1 – Analyse des gaps.

La relaxation lagrangienne nous donne de bonnes bornes MIN (GLg). En effet, et d'autant plus si nous comparons avec les valeurs des gaps obtenus de la relaxation fractionnaire (GLP), on peut voir que la borne MIN est proche du coût de la solution optimale.

Le mécanisme de projection fonctionne la plupart du temps. S'il ne parvient pas à répondre aux demandes à la première réplication, il obtient tout de même satisfaction lors des suivantes.

### 8.4.2 Expérimentations avec résolution heuristique FACLOC

Quand de larges instances sont considérées, des procédures heuristiques sont utilisées pour résoudre le problème de localisation. Il est intéressant d'étudier dans ce cas, la variation de la qualité des solutions par rapport aux méthodes exactes.

La FACLOC/heuristique est une méthode *randomisée* GRASP. Une première solution est obtenue en considérant que chaque demande liée à une usine sera satisfaite par cette usine. Vient ensuite un calcul du plus court chemin entre les différentes usines/périodes, qui permet de connaître le coût de circulation des produits entre les usines/périodes. Ces valeurs permettent ensuite de fusionner et échanger les différents nœuds de production pour améliorer la solution initiale. Il peut être intéressant d'utiliser un algorithme plus sophistiqué comme un de ceux décrits par Resende and Werneck (2004).

On utilise les paramètres suivants :

— Ite-Approx (resp. Def-Approx) représente le paramètre Ite (resp. Def) défini en section 8.4.1, pour le cas où le problème de localisation est résolu de manière heuristique,
— Gap-Approx et GLg-Approx sont les taux d'augmentation (en %) des valeurs, respectivement, Gap et GLg, selon les expérimentations conduites en section 8.4.1, pour le cas où le problème de localisation est résolu de manière heuristique,
— Gap-Loc-Approx est l'erreur moyenne (en %) induite par l'utilisation de l'heuristique FACLOC au lieu d'un *solveur* au moment de la résolution du sous-problème de localisation.

Nous réalisons une analyse de sensibilité dont les résultats sont donnés en tableau 8.2.

| Size | Ite-Approx | Gap-Approx (%) | GLg-Approx (%) | Gap-Loc-Approx (%) | Def-Approx (%) |
|---|---|---|---|---|---|
| (3,5,3) | 2 | 10.8 | 6.7 | 6.5 | 2.3 |
| (3,5,3) | 1 | 2.0 | 2.0 | 2.1 | 0 |
| (3,5,3) | 1 | 12.6 | 12.8 | 12.5 | 0 |
| (4,10,6) | 2 | 9.9 | 5.1 | 3.8 | 1.8 |
| (4,10,6) | 1 | 6.8 | 6.7 | 6.4 | 0 |
| (4,10,6) | 1 | 0.1 | 1.5 | 1.9 | 0 |
| (5,10,10) | 2 | 11.0 | 9.6 | 8.5 | 5.2 |
| (5,10,10) | 2 | 20.7 | 15.0 | 13.4 | 2.1 |
| (5,10,10) | 1 | 8.4 | 6.9 | 7.7 | 0 |
| (8,10,15) | 1 | 5.0 | 5.1 | 4.8 | 0 |
| (8,10,15) | 1 | 14.5 | 16.4 | 15.2 | 0 |
| (8,10,15) | 3 | 10.2 | 4.5 | 3.9 | 6.2 |
| (10,15,15) | 1 | 12.5 | 14.9 | 10.6 | 0 |
| (10,15,15) | 1 | 7.0 | 6.8 | 7.3 | 0 |
| (10,15,15) | 2 | 3.2 | 7.6 | 5.5 | 2.8 |

TABLE 8.2 – Analyse de sensibilité : heuristique / méthode exacte pour FACLOC.

On peut voir que l'augmentation de GLg-Approx suit de près le gap entre la solution optimale et le résultat fourni par l'heuristique.

Pourtant, l'impact sur la valeur du gap final est plus difficile à prévoir, en raison de la dépendance du comportement du mécanisme de projection sur l'ensemble des arcs de production en fin de la procédure LAG-LOT-SIZING.

### 8.4.3 Expérimentations avec résolution exacte du problème de localisation - partie 2

Comme en section 8.4.1, on résout le problème de localisation de manière exacte mais en générant cette fois-ci un autre type d'instances avec $R_{cp}(k) = 0, k \in K$. Le tableau 8.3 reporte les résultats de trois ensembles de 5 instances.

| Taille | Ite | Gap (in %) | GLg (in %) | GLP (in %) | GFr (in %) |
|---|---|---|---|---|---|
| (3,5,6) | 2 | 2,01 | 4,90 | 3,54 | 52,16 |
| (3,5,6) | 2 | 1,29 | 3,94 | 3,93 | 30,16 |
| (3,5,6) | 2 | 0,85 | 6,45 | 2,28 | 34,47 |
| (3,5,6) | 2 | 1,42 | 3,70 | 3,70 | 48,22 |
| (3,5,6) | 2 | 0,00 | 10,51 | 1,28 | 62,46 |
| (3,3,10) | 1 | 2,13 | 5,76 | 6,50 | 57,02 |
| (3,3,10) | 2 | 0,00 | 6,56 | 5,08 | 62,74 |
| (3,3,10) | 2 | 0,98 | 6,37 | 6,29 | 94,34 |
| (3,3,10) | 2 | 1,69 | 18,57 | 6,22 | 112,65 |
| (3,3,10) | 2 | 0,11 | 8,04 | 3,63 | 65,13 |
| (5,5,15) | 2 | 4,40 | 6,41 | 18,16 | 16,71 |
| (5,5,15) | 2 | 2,71 | 4,25 | 12,34 | 18,49 |
| (5,5,15) | 2 | 5,04 | 6,20 | 18,88 | 16,71 |
| (5,5,15) | 2 | 4,38 | 5,69 | 15,35 | 20,19 |
| (5,5,15) | 2 | 2,44 | 3,71 | 13,00 | 20,63 |

TABLE 8.3 – Analyse des gaps.

Dans le tableau 8.3, on note que, même avec une capacité de production faible, la procédure proposée résout efficacement le problème. La solution optimale est même atteinte pour certaines instances. Cependant, les gaps obtenus ont augmenté en comparaison avec le tableau 8.1, mais sont, au plus, égaux à 5,04. L'impact des contraintes de capacité peut aussi être mesuré par une forte augmentation des valeurs GFr.

Pour ces différentes expérimentations, les temps CPU pour les instances $(5, 10, 10)$ ne prennent pas plus d'une minute en moyenne pour l'heuristique FACLOC, alors qu'il a fallu jusqu'à 10 minutes pour les instances $(10, 15, 15)$. Ces temps obtenus avec l'utilisation de l'heuristique sont très liés aux paramètres de cette dernière et principalement au nombre d'itérations.

Au niveau de la procédure FACLOC appelant CPLEX, il est clair que la résolution du problème de localisation prend plus de temps. En effet, 60 minutes sont nécessaires pour obtenir une solution d'une instance de taille $(10, 15, 15)$.

Une étude plus détaillée de ces temps devrait permettre un choix plus fin de l'heuristique utilisée dans le problème de localisation ainsi que celui résolvant le problème de multi-flots. La principale contribution était ici de montrer que la décomposition proposée amenait à l'obtention de bonnes solutions.

## Perspectives et conclusion sur le *Lot-Sizing*

Nous venons de décrire une décomposition du problème de *Lot-Sizing* multi-usines, multi-périodes et multi-produits avec capacités de production, de transfert et de stockage. Nous le décomposons en problèmes de localisation et de multi-flots. Pour le premier, nous relâchons l'ensemble des contraintes de capacité afin de sélectionner plus rapidement un groupe de producteurs introduit ensuite dans le second (avec retour des contraintes de capacités). Nous reformulons le problème (mise à jour de la demande et des capacités) et relançons ce schéma global tant que la demande n'est pas entièrement satisfaite. Nous avons longuement étudié cette technique afin de maximiser ses performances et également sa robustesse. Les tests ont permis de révéler différents cas particuliers comme l'épuisement de tous les producteurs dans les premières périodes ne permettant plus de satisfaire la demande résiduelle à ces dates. L'évaluation de nos heuristiques révèle de bonnes performances malgré la complexité des instances générées.

Selon le même schéma, nous pourrions traiter d'un problème encore plus difficile intégrant des coûts fixes sur les transferts et le stockage, correspond aux frais de préparation. La relaxation lagrangienne aidant, telle qu'elle est déjà écrite, ce ne serait pas seulement l'ensemble des noeuds de production qui seront sélectionnés par la localisation mais aussi les arcs de transferts et stockage formant ensuite le graphe du problème de multi-flots.

# Conclusion Générale

Les chapitres 3, 4, et 5 traitent de 3 problèmes de *Dial-a-Ride* : le DARP classique, le DARP avec division de charge (DARPSL) et le DARP avec transferts (DARPT). Dans le premier des trois, un modèle et le *framework* associé sont réalisés de manière suffisamment générique pour pouvoir s'ouvrir aux deux problèmes préemptifs. La résolution des trois problèmes est effectuée en partie grâce à la propagation des contraintes de temps, c'est-à-dire les contraintes les plus difficiles à traiter (chaque demande implique deux fenêtres de temps, une durée maximum d'acheminement et chaque tournée est bornée d'une durée maximum). Au fil des insertions, les fenêtres se contractent de façon à ne laisser que l'espace temps satisfaisant ces contraintes. Seule celle de capacité, plus facile à gérer, est au préalable testée de manière naturelle. La propagation est élargie à partir du moment où les transferts sont possibles. En effet, chaque demande satisfaite en deux étapes reliées par transbordement implique des contraintes de temps à respecter sur deux tournées. Au bout d'un certain nombre d'insertions de ce type, une seule demande peut impliquer une contraction de fenêtres sur l'ensemble des tournées. Ces trois chapitres ont fait l'objet d'un nombre important de tests et, pour les deux cas préemptifs, des instances ont été générées selon des scénarios bien précis.

Le chapitre 6 introduit la notion de robustesse dans les travaux de *Dial-a-Ride*. Partant du constat que chaque insertion concerne les tournées courantes et non la place laissée aux futures demandes, il présente une mesure de ce que nous avons appelé l'*Insérabilité*. Celle-ci porte sur la contraction des fenêtres de temps suite à la l'insertion d'une demande et donc à la propagation des contraintes les plus difficiles. Son utilisation dans un contexte dynamique est étudiée en anticipant les demandes futures. Ainsi, les demandes sont insérées de façon à laisser suffisamment de place aux futures. De plus, ce même chapitre se pose la question des rendez-vous. En effet, une fois ces derniers fournis aux usagers, les tournées contractent les fenêtres de temps de façon à ce qu'elles soient d'amplitude nulle ou égale aux écarts supportés par les clients. Cette faible amplitude implique des tournées très peu flexibles dans un contexte de transport à la demande.

Le chapitre 7 a présenté un nouveau problème de la recherche opérationnelle permettant aux véhicules automatiques tels que les VIPA d'être supervisés. La technologie actuelle des véhicules implique leur utilisation sur circuit fermé. La minimisation des arrêts est alors apparue comme le meilleur objectif à atteindre pour satisfaire les contraintes de fiabilité. Indirectement, un faible nombre d'arrêts implique un gain de temps (les opérations de sorties de routes, chargement et déchargement sont

longues), une plus faible usure des véhicules et une baisse de l'énergie dépensée (les opérations de décélération, d'ouverture/fermeture des portes et d'accélération). Nous avons adapté ce problème en utilisant les principales contraintes temporelles du DARP (fenêtre de temps et durée maximum d'acheminement afin de limiter le nombre de tours) et testé plusieurs des solutions proposées.

Le chapitre 8 traite d'un problème de *Lot-Sizing* difficile. Il consiste à minimiser la somme des coûts fixes et variables de production et d'autres coûts variables pour le transfert et le stockage. Plusieurs types de produits sont envisagés et la production peut se faire sur plusieurs périodes et plusieurs sites tout comme la demande qui peut donc être différente en temps et en espace. Nous avons enfin ajouté des capacités aux opérations de production, stockage et transfert. Le problème est traité par une double décomposition : un premier problème (maître) de localisation obtenu par la relaxation lagrangienne des contraintes de capacité puis, une fois les sites producteurs sélectionnés, un second problème (esclave) de multiflots relaxant la décision d'ouverture des sites de production puisque ces derniers on déjà été sélectionnés par le processus précédent. Les expérimentations menées ont montré la qualité des résultats obtenus par cette approche et en mettant en valeur l'importance du choix de la procédure de localisation.

Ce mémoire a tenté de résumer les travaux réalisés le long de ce doctorat. Une bonne partie traite de problèmes nouveaux ouvrant donc sur un grand nombre de perspectives que nous avons relevées dans les conclusions de chaque chapitre. Que ce soit à propos du problème de la minimisation des arrêts permettant aux flottes de véhicules autonomes tels qu'ils sont utilisés aujourd'hui ou bien de l'optimisation de l'*Insérabilité* pour un transport à la demande plus robuste, beaucoup de nouvelles études sont à envisager.

Enfin, l'évolution de la technologie des véhicules sans conducteur amènera des modifications de contraintes et il s'ensuivra de nouvelles problématiques de recherche opérationnelle. Les premières exploitations des VIPA, par exemple, sur le site de Ladoux de Michelin, permettront d'obtenir des instances concrètes en donnant ainsi la possibilité d'ajuster nos problèmes aux différents flux de demandes : ce sera l'occasion de parfaire nos algorithmes et d'affiner nos instances. Nous aurons, par ce biais, la possibilité d'avoir des flux dynamiques qui pourront alors être modélisés, ce qui permettra une utilisation concrète de l'*Insérabilité* avec des demandes futures basées sur ce modèle avec ainsi la possibilité de les anticiper.

# Acronymes

**DARP :** *Dial-a-Ride Problem.*

**DARPSL :** *Dial-a-Ride Problem with Split Loads.*

**DARPT :** *Dial-a-Ride Problem with transfers.*

**ISIMA :** Institut Supérieur d'Informatique de Modélisation et de leurs Applications.

**ITS :** *Intelligent Transport System.*

**LIMOS :** Laboratoire d'Informatique, Modélisation et Optimisation des Systèmes.

**LSP :** *Lot Sizing Problem.*

**ODT :** *On Demand Transportation.*

**PDP :** *Pickup and Delivery Problem.*

**PDPTW :** *Pickup and Delivery Problem with Time Windows.*

**PLNE :** Programme linéaire en nombres entiers.

**TAD :** Transport à la demande.

**TUM :** *Totally Unimodular Matrix.*

**TIMS :** Technologies de l'information de la mobilité et de la sûreté.

**VIPA :** Véhicule Individuel Public Autonome.

**VRP :** *Vehicle Routing Problem.*

# Bibliographie

ADASE (2000). Advanced driver assistance systems in europe (adase). `www.adase2.net/`, 2000.

ADEME (2010). Caractérisation de services et usages de covoiturage en france : Quels impacts sur l'environnement, quelles perspectives d'amélioration. `www2.ademe.fr/`, 2010.

AHSRA (1989). Advanced cruise-assist highway system research association. `www.ahsra.or.jp/`, 1989.

Aksen, D., Altınkemer, K., and Chand, S. (2003). The single-item lot-sizing problem with immediate lost sales. *European Journal of Operational Research*, 147(3) :558–566.

Analysis, N. (2000). User needs analysis and analysis of key technologies. *Report on User Needs for Cybernetic Transport System (CTS)*.

Archetti, C. and Speranza, M. (2008). The vehicle routing problem : Latest advances and new challenges, chapter the split delivery vehicle routing problem : A survey. *pages 103–122. Springer US*.

Archetti, C., Speranza, M., and Hertz, A. (2006). A tabu search algorithm for the split delivery vehicle routing problem. *Transportation Science, 40(1) : 64–73*.

Ascher, F. (2002). Le transport à la demande : individualisation des mobilités urbaines et personnalisation des services publics. *Annales des télécommunications 57(3-4), 277-288*.

Attanasio, A., Cordeau, J., Ghiani, G., and Laporte, G. (2004). Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing. Volume 30, Issue 3, 377-387*.

Autolib (2011). Autolib. `https://www.autolib.eu/`, 2011.

Avego (2013). Service de covoiturage dynamique. *https ://rtr.avego.com/*.

Barany, I., Van Roy, T. J., and Wolsey, L. A. (1984). Strong formulations for multi-item capacitated lot sizing. *Management Science*, 30(10) :1255–1261.

Beaudry, A., Laporte, G., Melo, T., and Nickel, S. (2012). Dynamic transportation of patients in hospitals. *OR Spectrum, Volume 32, Issue 1, 77-107*.

Berbeglia, G., Cordeau, J., Gribkovskaia, I., and Laporte, G. (2007). Static pickup and delivery problems : a classification scheme and survey. *Journal of the Spanish Society of Statistics and Operations Research, 15, 1-31*.

Berbeglia, G., Cordeau, J.-F., and Laporte, G. (2012). A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, 24(3) :343–355.

Berger, T., Sallez, Y., Raileanu, S., Tahon, C., Trentesaux, D., and Borangiu, T. (2011). Personal rapid transit in an open-control framework. *Computers & Industrial Engineering*, 61(2) :300–312.

Bilge and Tanchoco (1997). Agv systems with multi-load carriers : Basic issues and potential benefits. *Journal of Manufacturing Systems, 16 (3) (1997), 159–174.*

Bodin, L. and Sexton, T. (1986). The multi-vehicle subscriber dial-a-ride problem. *TIMS Studies in Management Science 2 : 73-86.*

Borndörfer, R., Grötschel, M., Klostermeier, F., and Küttner, C. (1999). *Telebus Berlin : Vehicle scheduling in a dial-a-ride system.* Springer.

Bremen (2005). Cambio stadtauto bremen gmbh. car-sharing in bremen, a succes story to be extended. `www.innovations-transports.fr/IMG/pdf/eit044.pdf/`, 2005.

Calvo, R., Wolfler, and Colorni, A. (2007). An effective and fast heuristic for the dial-a-ride problem. *4OR*, 5(1) :61–73.

Castex, E. (2007). *Le transport à la demande (TAD) en France : de l'état des lieux à l'anticipation.* PhD thesis.

Centre d'Etudes sur les Réseaux, les Transports, l. e. l. c. p. (2009). Le covoiturage dynamique - Étude préalable avant expérimentation. `www.lara.inist.fr/bitstream/handle/2332/1463/CERTU-RE_09-03.pdf/`, 2009.

CERTU (2007). Le covoiturage en france et en europe. etat des lieux et perspectives. `www.innovations-transports.fr/IMG/pdf/covoiturage.pdf/`, 2007.

Chen, W.-H. and Thizy, J.-M. (1990). Analysis of relaxations for the multi-item capacitated lot-sizing problem. *Annals of operations Research*, 26(1) :29–72.

Chevrier, R. (2008). *Optimisation de transport à la demande dans des territoires polarisé.* PhD thesis.

Chevrier, R., Canalda, P., Chatonnay, P., and Josselin, D. (2006). Comparison of three algorithms for solving the convergent demand responsive transportation problem. *ITSC'2006, 9th Int. IEEE Conf. on IntelligentTransportation Systems, Toronto, Canada, 1096–1101.*

CityMobil (2011). `www.citymobil-project.eu/`, 2011.

Clay (2000). Millennium problems : P vs np problem. `http://www.claymath.org/millenium-problems/p-vs-np-problem`, 2000.

Colorni, A. and Righini, G. (2001). Modeling and optimizing dynamic dial-a-ride problems. *International Transactions in Operational Research - Volume 8, Issue 2, pages 155–166, March 2001.*

Communauto (2013). `www.communauto.com/`, 2013.

Cook, S. (1971). The complexity of theorem proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing, 151-158.*

Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3) :573–586.

Cordeau, J.-F. and Laporte, G. (2003). A tabu search heuristic algorithm for the static multi-vehicle dial-a-ride problem. *Transportation Research B 37, 579–594.*

Cordeau, J.-F. and Laporte, G. (2007). The dial-a-ride problem : models and algorithms. *Annals of Operations Research*, 153(1) :29–46.

Cortés, C. E. and Jayakrishnan, R. (2002). Design and operational concepts of high-coverage point-to-point transit system. *Transportation Research Record : Journal of the Transportation Research Board*, 1783(1) :178–187.

Cortes, C. E., Matamala, M., and Contardo, C. (2010). The pickup and delivery problem with transfers. *Formulation and a branch-and-cut solution method, European Journal of Operational Research 200, 711-724.*

Coslovich, L., Pesenti, R., and Ukovich, W. (2006). A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research 175, 1605–1615.*

Covoiturage.fr (2006). `www.covoiturage.fr/`, 2006.

Dantzig, G. B. (1982). Reminiscences about the origins of linear programming. *Operations Research Letters*, 1(2) :43–48.

Deleplanque, S. (2012). Heuristiques rapides pour des problèmes de transport à la demande. *Journées scientifiques de l'école doctorale des sciences pour l'ingénieur - UBP Clermont-Ferrand.*

Deleplanque, S. (2013). Vers le transport à la demande partagé autonome. *Auvergne Sciences (journal de la science en auvergne).*

Deleplanque, S., Derutin, J.-P., and Quilliot, A. (2013a). Anticipation in the dial-a-ride problem : an introduction to the robustness. In *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, pages 299–305. IEEE.

Deleplanque, S., Duhamel, C., Kedad-Sidhoum, S., Liberalino, H., and Quilliot, A. (2012a). Décomposition d'un problème de lot-sizing en problèmes de localisation et de multiflots. *ROADEF 2012, Anger.*

Deleplanque, S., Kedad-Sidhoum, S., and Quilliot, A. (2012b). Lagrangean based lower bounds for a multi-plant lot-sizing problem with capacity constraints. *CO 2012 - International Symposium on Combinatorial Optimization 2012, Oxford (UK).*

Deleplanque, S., Kedad-Sidhoum, S., and Quilliot, A. (2012c). Lagrangean decomposition of a lot-sizing problem into facility location and multi-commodity flow. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*, pages 385–392. IEEE.

Deleplanque, S., Kedad-Sidhoum, S., and Quilliot, A. (2013b). Lagrangean heuristic for a multi-plant lot-sizing problem with transfer and storage capacities. *RAIRO-Operations Research*, 47(04) :429–443.

Deleplanque, S., Marques, A., Bernardes Real, L., and Quilliot, A. (2013c). Le problème de la minimisation des arrêts. *ROADEF 2013, Troyes.*

Deleplanque, S. and Quilliot, A. (2012a). Decomposing a multi-plant, multi-item lot sizing problem with transfer costs/capacities into facility location and flow subproblems. In *Information Control Problems in Manufacturing*, volume 14, pages 1433–1438.

Deleplanque, S. and Quilliot, A. (2012b). Insertion techniques and constraint propagation for the darp. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*, pages 393–400. IEEE.

Deleplanque, S. and Quilliot, A. (2013a). Constraint propagation for the dial-a-ride problem with split loads. In *Recent Advances in Computational Optimization*, pages 31–50. Springer.

Deleplanque, S. and Quilliot, A. (2013b). Dial a ride problem avec transferts et division du chargement. *ROADEF 2013, Troyes.*

Deleplanque, S. and Quilliot, A. (2013c). Dial-a-ride problem with time windows, transshipments, and dynamic transfer points. In *Manufacturing Modelling, Management, and Control*, volume 7, pages 1256–1261.

Deleplanque, S. and Quilliot, A. (2013d). Introduction à la notion d'anticipation et de robustesse dans les problèmes de dial-a-ride dynamiques. *JDJN MACS 2013, Strasbourg.*

Deleplanque, S. and Quilliot, A. (2013e). Robustesse dans le darp dynamique : anticipation des demandes. *ROADEF 2013, Troyes.*

Deleplanque, S. and Quilliot, A. (2013f). Transfers in the on-demand transportation : the darpt. In *Proceedings of the Multidisciplinary International Scheduling Conference : Theory and Applications (MISTA)*, number 2013, pages 185–205.

Deleplanque, S. and Quilliot, A. (2014a). Handling uncertain demands in the dynamic dial-a-ride problem. *20th Conference of the International Federation of Operational Research Societies - IFORS.*

Deleplanque, S. and Quilliot, A. (2014b). Introduction à la notion d'anticipation et de robustesse dans les problèmes de dial-a-ride dynamiques. *Journal Européen des Systèmes Automatisés / European Journal of Automation*, 48.

Deleplanque, S. and Quilliot, A. (2014c). Robustness tools in dynamic darp. *Recent Advances in Computational Optimization. Studies in Computational Intelligence.*

Deleplanque, S., Quilliot, A., and Toussaint, H. (2012d). Décomposition d'un problème de lot-sizing en problèmes de localisation et de multiflots. *ROADEF 2012, Anger.*

Demaine, E., Hohenberger, S., and Liben-Nowell, D. (2003). Tetris is hard, even to approximate. *COCOON 2003.*

Desrosiers, J., Dumas, Y., Solomon, M., and Soumis, F. (1995). Time constrained routing and scheduling. *Volume 8 dans Handbooks in Operations Research and Management Science, 35–139. Amsterdam, North-Holland : Informs.*

Desrosiers, J., Dumas, Y., and Soumis, F. (1986). A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3) :301–326.

Desrosiers, J., Dumas, Y., Soumis, F., Taillefer, S., and Villeneuve, D. (1991). An algorithm for mini-clustering in handicapped transport. *Cahiers du GERAD.*

Dewen, Z., Li, J., Yuwen, Z., Guanghui, S., and Kai, H. (1997). Modern elevator group supervisory control systems and neural networks technique. In *Intelligent Processing Systems, 1997. ICIPS'97. 1997 IEEE International Conference on*, volume 1, pages 528–532. IEEE.

Diana, M. and Dessouky, M. (2004). A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B : Methodological*, 38(6) :539–557.

Dresner, K. M. and Stone, P. (2008). A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res.(JAIR)*, 31 :591–656.

Drexl, A. and Kimms, A. (1997). Lot sizing and scheduling ?survey and extensions. *European Journal of Operational Research*, 99(2) :221–235.

Dror, M. and Trudeau, P. (1989). Savings by split delivery routing. *Transportation Science, 23(2) :141–145.*

Dumas, Y., Desrosiers, J., and Soumis, F. (1989). *Large scale multi-vehicle dial-a-ride problems.* Groupe d'études et de recherche en analyse des décisions.

Dupuy (1995). Les territoires de l'automobile. *Collection Villes. Anthropos.*

Dupuy, G. (1999). *La dépendance automobile : symptômes, analyses, diagnostic, traitements.* Anthropos.

Elmaghraby, S. E. (1978). The economic lot scheduling problem (elsp) : review and extensions. *Management Science*, 24(6) :587–598.

Falkenauer, E. and Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 1186–1192.

Feo, T. A. and Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2) :67–71.

Florian, M. and Klein, M. (1971). Deterministic production planning with concave costs and capacity constraints. *Management Science*, 18(1) :12–20.

Florian, M., Lenstra, J. K., and Rinnooy Kan, A. (1980). Deterministic production planning : Algorithms and complexity. *Management science*, 26(7) :669–679.

Folsom, T. C. (2011). An invitation to a new transportation mode. In *16th IASTED International Conference on Robotics and Applications.*

Fortnow, L. (2009). The status of the p versus np problem. *Communications of the ACM*, 52(9) :78–86.

Fraichard (2005). Navigation (paris). *Navigation (Paris), 53(209) :53-74.*

Fu, L. (2002). A simulation model for evaluating advanced dial-a-ride paratransit systems. *Transportation Research Part A : Policy and Practice*, 36(4) :291–307.

Fujino, A., Tobita, T., Segawa, K., Yoneda, K., and Togawa, A. (1997). An elevator group control system with floor-attribute control method and system optimization using genetic algorithms. *Industrial Electronics, IEEE Transactions on*, 44(4) :546–552.

Garaix, T. (2007). Etude de résolution exacte de problèmes de transport à la demande avec qualité de service. *Thèse de doctorat. Soutenue en décembre 2007.*

Garaix, T., Josselin, D., Feillet, D., Artigues, C., and Castex, E. (2007). Transport à la demande points à points en zone peu dense. proposition d'une méthode d'optimisation de tournées. *CyberGeo : European Journal of Geography.*

Glover, F. (1989 & 1990). Tabu search - part 1 (190-206) & 2 (4-32). *ORSA Journal on Computing 1 & 2.*

Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics 65, 223-253.*

Google (2010). *googleblog.blogspot.com/2010/10/what-were-driving-at.html.*

Graves, S., Kan, A. R., and Zipkin, P. (1993). Handbooks in operations research and management science, vol. 4 : logistics of production and inventory. *North-Holland.*

Grötschel, M., Hauptmeier, D., Krumke, S., and Rambau, J. (1999). *Simulation studies for the online Dial-a-Ride problem.* Konrad-Zuse-Zentrum für Informationstechnik Berlin.

Healy, P. and Moll, R. (1995). A new extension of local search applied to the dial-a-ride problem. *European Journal of Operational Research 83, 83–104.*

Hertz, A. and Widmer, M. (2003). Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational Research*, 151(2) :247–252.

Igarashi, K., Take, S., and Ishikawa, T. (1994). Supervisory control for elevator group with fuzzy expert system. In *Industrial Technology, 1994., Proceedings of the IEEE International Conference on*, pages 133–137.

Igocars (2013). `I-GO.www.igocars.org/`, 2013.

Ioachim, I., Desrosiers, J., Dumas, Y., Solomon, M., and Villeneuve, D. (1995). A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science*, 29(1) :63–78.

IVI (1998). Intelligent vehicle initiative. `www.its.dot.gov/ivi/ivi.htm/`, 1998.

Jans, R. and Degraeve, Z. (2008a). Modeling industrial lot sizing problems : a review. *International Journal of Production Research*, 46(6) :1619–1643.

Jans, R. and Degraeve, Z. (2008b). Modeling industrial lot sizing problems : a review. *International Journal of Production Research*, 46(6) :1619–1643.

Jaw, J., Odoni, A., Psaraftis, H., and Wilson, N. (1986). A heuristic algorithm for the multi-vehicle many-to-many advance request dial-a-ride problem. *Transportation Research B 20B, 243–257.*

Jorgensen, R., Larsen, J., and Bergvinsdottir, K. (2007). Solving the dial-a-ride problem using genetic algorithms. *Journal of the Operational Research Society, 58(10) :1321-1331.*

Karp, R. (1972). Reducibility among combinatorial problems. *RE. Miller and JW. Thatcher (editors). Complexity of Computer Computations. New York : Plenum (éditeurs). 85–103.*

Kedad-Sidhoum, S., Rodríguez-Getán, C., Absi, N., and Dauzère-Pérès, S. (2009). Problème de lot-sizing à capacité finie avec fenêtres de temps et contraintes de services. *ROADEF 2009 Nancy.*

Kergosiena, Y., Lentéa, C., Pitonb, D., and Billauta, J. (2011). A tabu search heuristic for the dynamic transportation of patients between care units. *European Journal of Operational Research Volume 214, Issue 2, 16 October 2011, 442–452.*

Kerivin, H. L. M., Lacroix, M., and Mahjoub, A. R. (2008). The splittable pickup and delivery problem with reloads. *European Journal of Industrial Engineering 2 (2), 112-133.*

Kim, J.-H. and Moon, B.-R. (2001). Adaptive elevator group control with cameras. *Industrial Electronics, IEEE Transactions on*, 48(2) :377–382.

Kirkpatrick, S., Vecchi, M., et al. (1983). Optimization by simmulated annealing. *science*, 220(4598) :671–680.

Kwon, O., Lee, E., and Bahn, H. (2014). Sensor-aware elevator scheduling for smart building environments. *Building and Environment*, 72(0) :332 – 342.

Lawler, E., Lenstra, K., Rinnoy-Kan, A., and Schmoys, D. (1993). Sequencing and scheduling : algorithms and complexity, in s.c. graves. *Handbook of Operation*

*Research and Management Sciences, Vol 4 : Logistics of Production and Inventory, North-Hollond, 445-522.*

Le-Anh, T. and De Koster, M. (2006). A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1) :1–23.

Leclaire, P. (2009). *Heuristiques et métaheuristiques pour des problèmes d'optimisation combinatoire : problèmes d'équilibrage de lignes d'assemblage et problèmes de tournées de véhicules, Thèse de doctorat.*

LILE, L. (2007). Autopartage. *lilas-autopartage.fr/.*

Lu, Q. and Dessouky, M. (2006). A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175(2) :672–687.

Luo, Y. and Schonfeld, P. (2007). A rejected-reinsertion heuristic for the static dial-a-ride problem. *Transportation Research Part B : Methodological*, 41(7) :736–755.

Madsen, O., Ravn, H., and Rygaard, J. (1995). A heuristic algorithm for the a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research 60, 193-208.*

Maes, J. and Van Wassenhove, L. (1988). Multi-item single-level capacitated dynamic lot-sizing heuristics : A general review. *Journal of the Operational Research Society*, pages 991–1004.

Masson, R., Lehuédé, F., and Péton, O. (2011). A tabu search algorithm for the dial-a-ride problem with transfers. *Proceedings of the International Conference on Industrial Engineering and Systems Management.*

Masson, R., Lehuédé, F., and Péton, O. (2012). Simple temporal problems in route scheduling for the dial–a–ride problem with transfers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 275–291. Springer.

Masson, R., Lehuédé, F., and Péton, O. (2013). An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science*, 47(3) :344–355.

MdN (2003). Modulobus de noël, rapport d'expérimentation. `http://www.innovations-transports.fr/IMG/pdf/resultat_d_experimentation_modulobus_-_noel_-1.pdf/`, 2003.

Meer, V. (2000). Operational control of internal transport system. *PhD thesis, Erasmus University, Rotterdam.*

Meer, V. and Koster, D. (1999). Using multiple load vehicles for internal transport with batch arrivals of loads. *M. Grazia Speranza, P. Stähly (Eds.), Advances in Distribution Logistics, Springer, Berlin (1999), pp. 197–214.*

Menger, K. (1932). Das botenproblem. *Ergebnisse eines mathematischenkolloquiums 2, 11–12.*

Minoux, M. (1983). *Programmation mathématique : théorie et algorithmes*, volume 1. Dunod Paris.

Mirchandani, P. B. and Francis, R. L. (1990). *Discrete location theory.*

Mitrović-Minić, S., Krishnamurti, R., and Laporte, G. (2004). Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B : Methodological*, 38(8) :669–685.

Mitrovic-Minic, S. and Laporte, G. (2006). The pickup and delivery problem with time windows and transshipment. *INFOR 44 (3) 217-228*.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11) :1097–1100.

Modulobus (2008). `www.agence-nationale-recherche.fr/projet-anr/?tx_lwmsuivibilan_pi2%5BCODE%5D=ANR-07-TSFA-0012/`, 2008.

Moovicite, C. d. M. d. l. C. (2013). `www.moovicite.com/index.php/transports-a-la-demande/`, 2013.

Nakao, Y. and Nagamochi, H. (2012). Worst case analysis for pickup and delivery problems with transfer. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E91-A (9) 2328-2334*.

Noda, I., Ohta, M., Shinoda, K., Kumada, Y., and Nakashima, H. (2003). Evaluation of usability of dial-a-ride systems by social simulation. *Multi-Agent-Based Simulation III*, pages 167–181.

Nowak, M., Özlem, E., and White, C. (2008). Pickup and delivery with split loads. *Transportation Science, 42(1) :32–43*.

Paquette, J., Cordeau, J., and Laporte, G. (2007). Etude comparative de divers modèles pour le problème de transport à la demande. *Rapport technique. CIRRELT, HEC Montréal*.

Parragh, S. (2011). Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Research Part C : Emerging Technologies*, 19(5) :912–930.

Parragh, S., Doerner, K., and Hartl, R. (2010). Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research, 37 1129–1138*.

Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1) :1 – 11.

Pochet, Y. and Wolsey, L. (2006). Production planning by mixed integer programming. *Springer Series in Operat. Res. And Financial Engineering*.

Potts, C. and Wassenhove, L. V. (1992). Integrated scheduling with batching and lot sizing : a review of algorithms and complexity . *Jour. Operat. Research Society 43, 395-406*.

Project, C. (2000). Safety standards for cybercars, part 1 : Existing standards and guidelines. `www.cybercars.org/docs/D6.1%20Guidelines.pdf/`, 2000.

Prometheus (1986). Programme for a european traffic system with highest efficiency and unprecedented safety. `www.eureka.be/`, 1986.

PrRoutiere (2012). Principaux facteurs d'accidents. `www.preventionroutiere.asso.fr/Nos-publications/Statistiques-d-accidents/Principaux-facteurs-d-accidents/`, 2012.

PRT, U. (2011). Urban light transport. `www.ultraprt.com/applications/existing-systems/`, 2011.

Psaraftis and Harilaos (1980). A dynamic programming approach for sequencing groups of identical jobs. *Operations Research*, 28(6) :1347–1359.

Psaraftis, H. (1983). An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science 17, 351–357*.

Psaraftis, H. N. (1995). Dynamic vehicle routing : Status and prospects. *annals of Operations Research*, 61(1) :143–164.

Quilliot, A., Deleplanque, S., and Bernay, B. (2014). Branch and price for a reliability oriented darp model. *ISCO, Lisbon.*

Rekiek, B., Lit, P., Pellichero, F., Eglise, T., Fouda, P., Falkenauer, E., and Delchambre, A. (2001). A multiple objective grouping genetic algorithm for assembly line design. *Journal of Intelligent Manufacturing*, 12(5-6) :467–485.

Resende, M. G. C. and Werneck, R. F. (2004). A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10(1) :59–88.

Routière, S. (2008). La sécurité routière à l'ordre du jour de l'onu. `www2.securiteroutiere.gouv.fr/vos-infos/presse/communiques/1-2008/CP_01-04-08.html/`, 2008.

Sambasivan, M. and Yahya, S. (2005). A lagrangean-based heuristic for multi-plant, multi-item, multi-period capacitated lot-sizing problems with inter-plant transfers. *Computers & Operations Research*, 32(3) :537–555.

Sato (1991). Overview of the advanced safety vehicle project and related activities. *Japan Automobile Research Institute Research Journal, 25(8).*

Schneider (1990). Automated highway systems research on design, development and implementation issues. `faculty.washington.edu/jbs/itrans/ahssummary.htm/`, 1990.

Shang, J. S. and Cuff, C. K. (1996). Multicriteria pickup and delivery problem with transfer opportunity. *Computers & Industrial Engineering*, 30(4) :631–645.

So, A., Beebe, J., Chan, W., and Liu, S. (1995). Elevator traffic pattern recognition by artificial neural network. *Elevator technology*, 6 :122–131.

SuperShuffle (2013). Veolia. `http://www.supershuttle.fr/`, 2013.

Tanaka, S., Uraguchi, Y., and Araki, M. (2005). Dynamic optimization of the operation of single-car elevator systems with destination hall call registration : Part ii. the solution algorithm. *European Journal of Operational Research*, 167(2) :574 – 587.

Teodorovic, D. and Radivojevic, G. (2000). A fuzzy logic approach to dynamic dial-a-ride problem. *Fuzzy sets and systems*, 116(1) :23–33.

Thangiah, S., Fergany, A., and Awam, S. (2007). Real-time split-delivery pickup and delivery time window problems with transfers. *Central European Journal of Operations Research - 15, 329-349.*

Toth, P. and Vigo, D. (1996). Fast local search algorithms for the handicapped persons transportation problem. In *Meta-Heuristics*, pages 677–690. Springer.

Trigeiro, W. W., Thomas, L. J., and McClain, J. O. (1989). Capacitated lot sizing with setup times. *Management science*, 35(3) :353–366.

Valejo, Meisner, Dias, and Nunes (2004). Cybernetic transport systems in coimbra, evaluation and demonstration for cybermove project. *Proc. of 2004 European Ele-Drive Tranportation.*

Velasco, N., Dejax, P., Guéret, C., and Prins, C. (2006). Un algorithme génétique pour un problème de collecte et livraison bi-objectif. *Actes de la 6ème Conférence Francophone de MOdélisation et de SIMulation, MOSIM06.*

Velib (2007). vélo et liberté, www.velib.paris.fr.

Wagner, H. M. and Whitin, T. M. (1958). Dynamic version of the economic lot size model. *Management science*, 5(1) :89–96.

Wong, K. and Bell, M. (2006). Solution of the dial-a-ride problem with multi-dimensional capacity constraints. *International Transactions in Operational Research*, 13(3) :195–208.

Zhang, Q., Manier, H., and Manier, M.-A. (2013). Metaheuristics for job shop scheduling with transportation. *Metaheuristics for Production Scheduling*, pages 465–493.